Winter Vacation Seminar & Paper Review

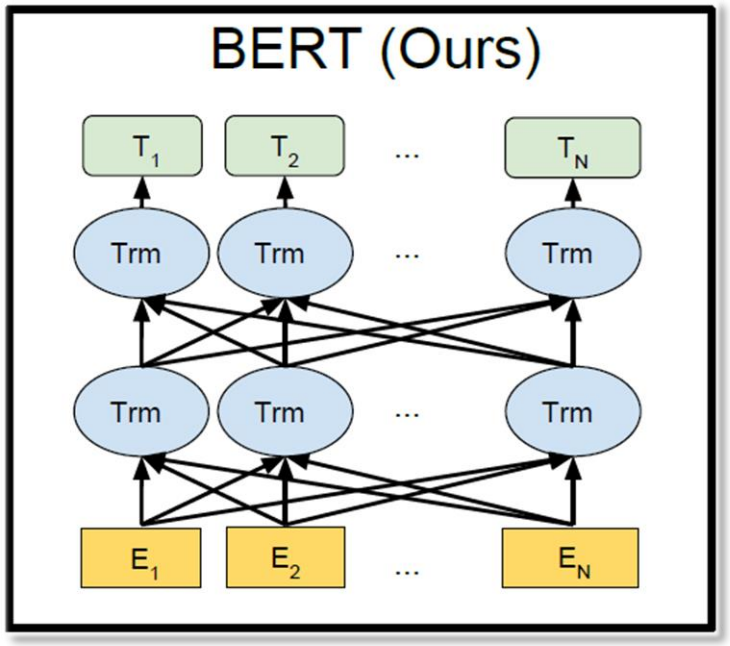
ELECTRA : PRETRAINING TEXT ENCODERS AS DISCRIMINATORS RATHER THAN GENERATORS

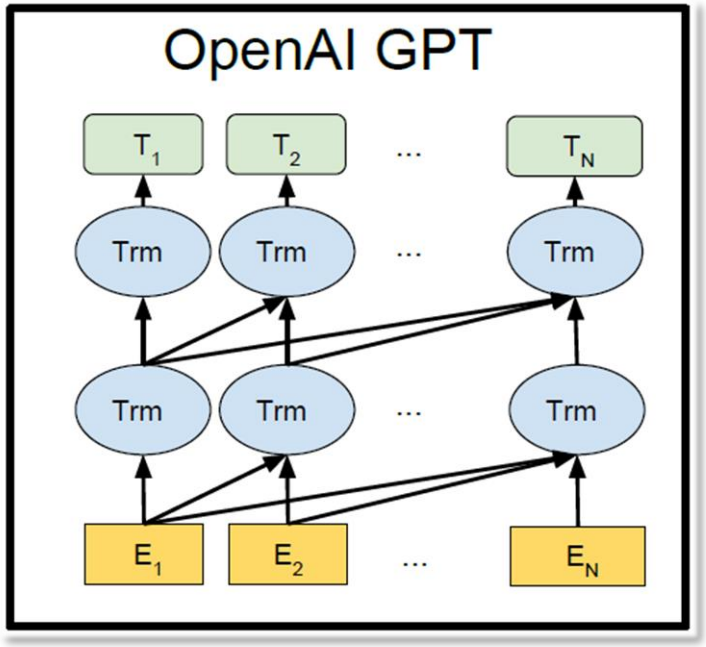
18th, January, 2021

Natural Language Learning Lab
Taegyeong, Eo

Pre-Trained Language Model

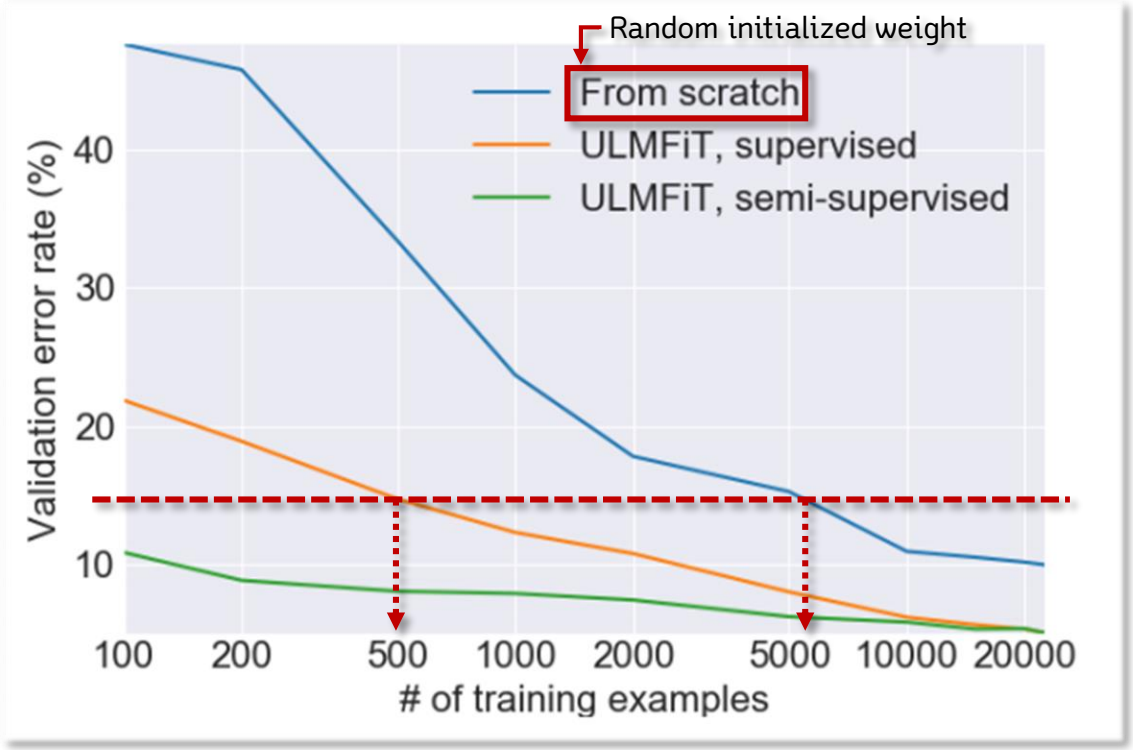
- ✓ 언어를 이해하도록 미리 학습된 모델
- ✓ Fine-tuning 하여 언어에 관련된 다양한 Task를 수행할 수 있음
  - Ex) Q/A, Sentence Classification, Named Entity Recognition …
- ✓ 크게 2가지 방식으로 분류
  - Bi-directional(BERT, ELMo), Uni-directional(GPT)

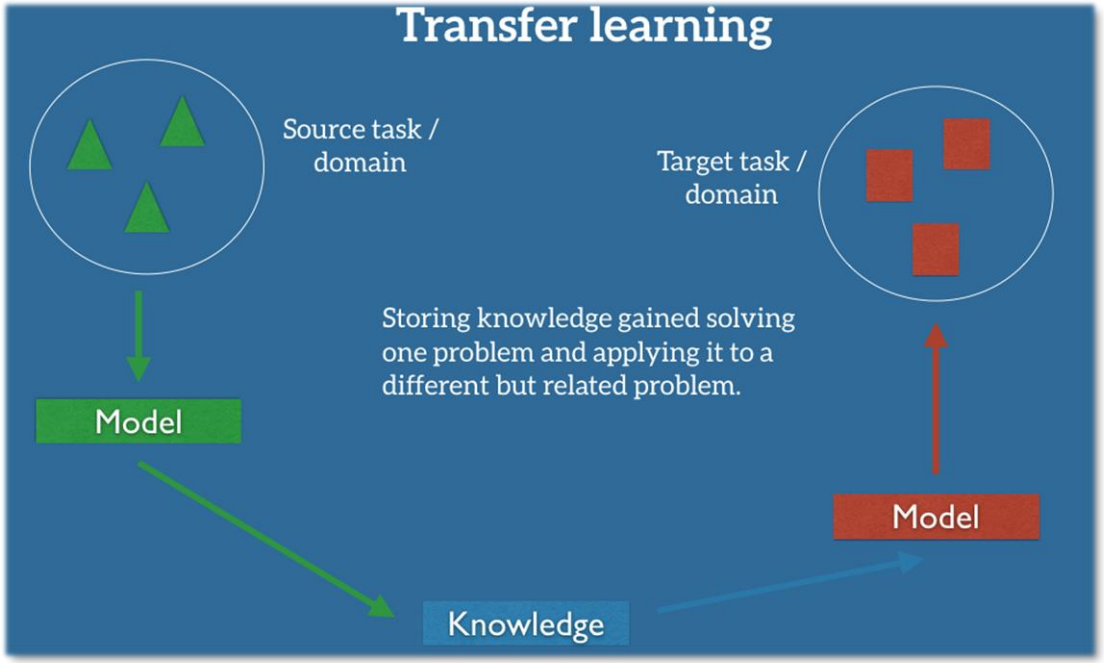




Transfer Learning

- ✓ 미리 학습된 모델의 특질(feature, knowledge)을 활용하여 새로운 모델에 적용하는 것
- ✓ 기존 모델의 Fine-tuning을 통해 Downstream Task를 수행
- ✓ Low-shot learning (※ Universal Language Model Fine-tuning for Text Classification)
  - 1/10의 데이터만 사용해서 동일한 성능





※ https://ruder.io/transfer-learning/index.html#whatistransferlearning

Fine-tuning

✓ Pre-trained Model을 변형하거나 레이어를 추가하여 새로운 데이터로 학습하는 것

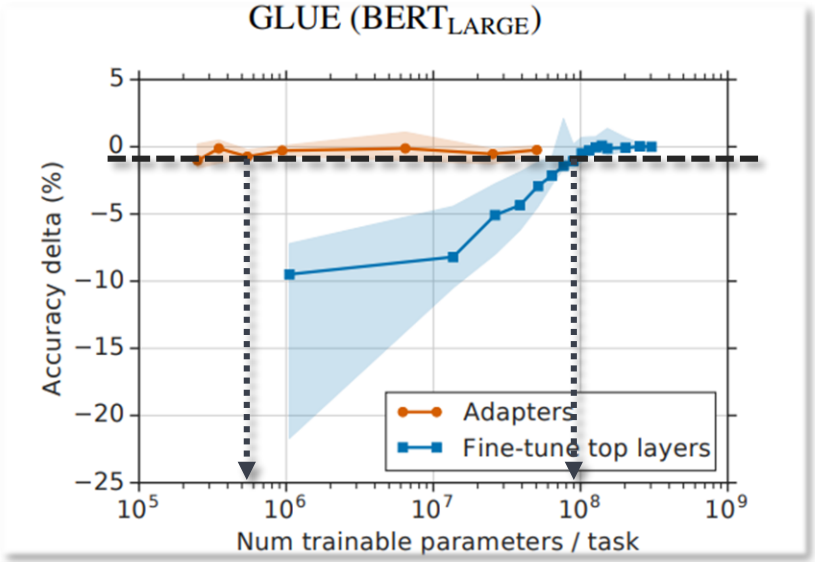
❖ 구조는 어떻게?

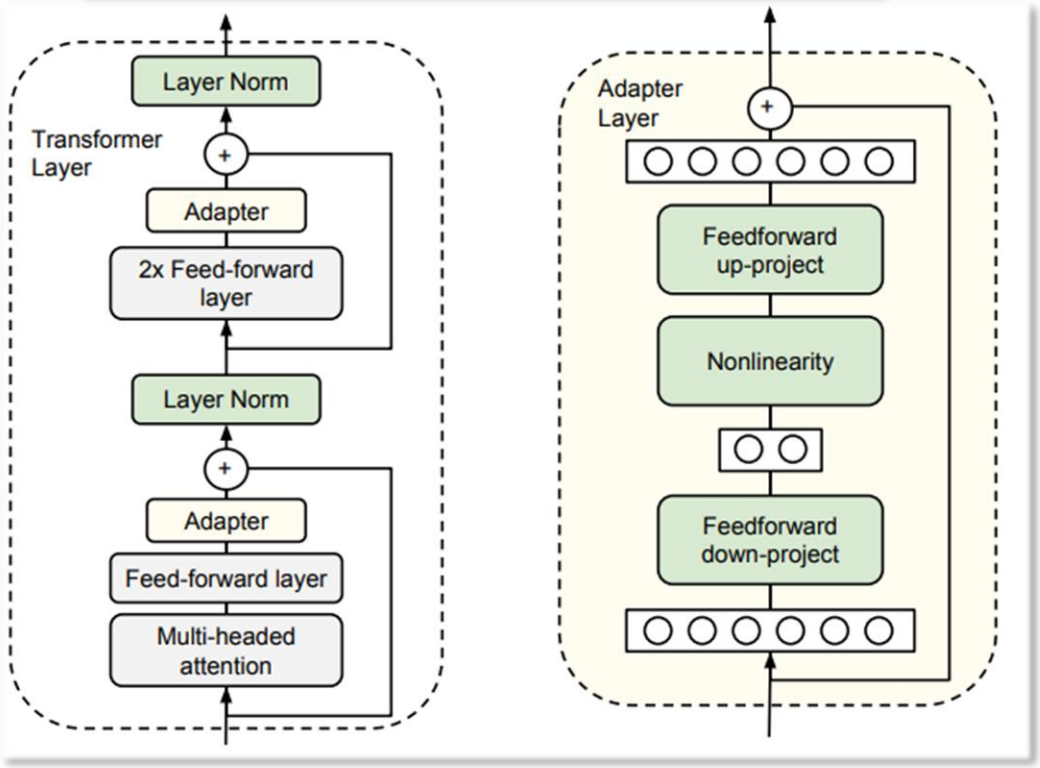
- 그대로 가져다 쓰는 경우
- 내부 구조를 변경하는 경우 (※ Parameter-Efficient Transfer Learning for NLP)
  - " Training adapters with sizes 0.5 – 5% of the original model, performance is within 1% of the competitive published results on $BERT_{LARGE}$ "

❖ 가중치는 어떻게?

- 가중치를 고정하는 경우
  - 가중치를 Freezing
- 가중치를 업데이트하는 경우
  - 기존 모델의 가중치로 초기화 후 학습

```
for param in model.bert.parameters():
    param.requires_grad = False
```





Masked Language Model (BERT)

- ✓ Input Sequence중 n%의 Token을 [MASK]로 치환
  - 보통 15%의 토큰을 치환하도록 설정
    - 15% ↑ : 문맥을 충분히 반영하지 못함
    - 15% ↓ : 학습 효율이 떨어짐
- ✓ 마스킹된 Sequence를 모델에 넣었을 때 원본을 맞추도록 학습
- ✓ Loss function

모델의 Output
차원이 vocab_size인 vector

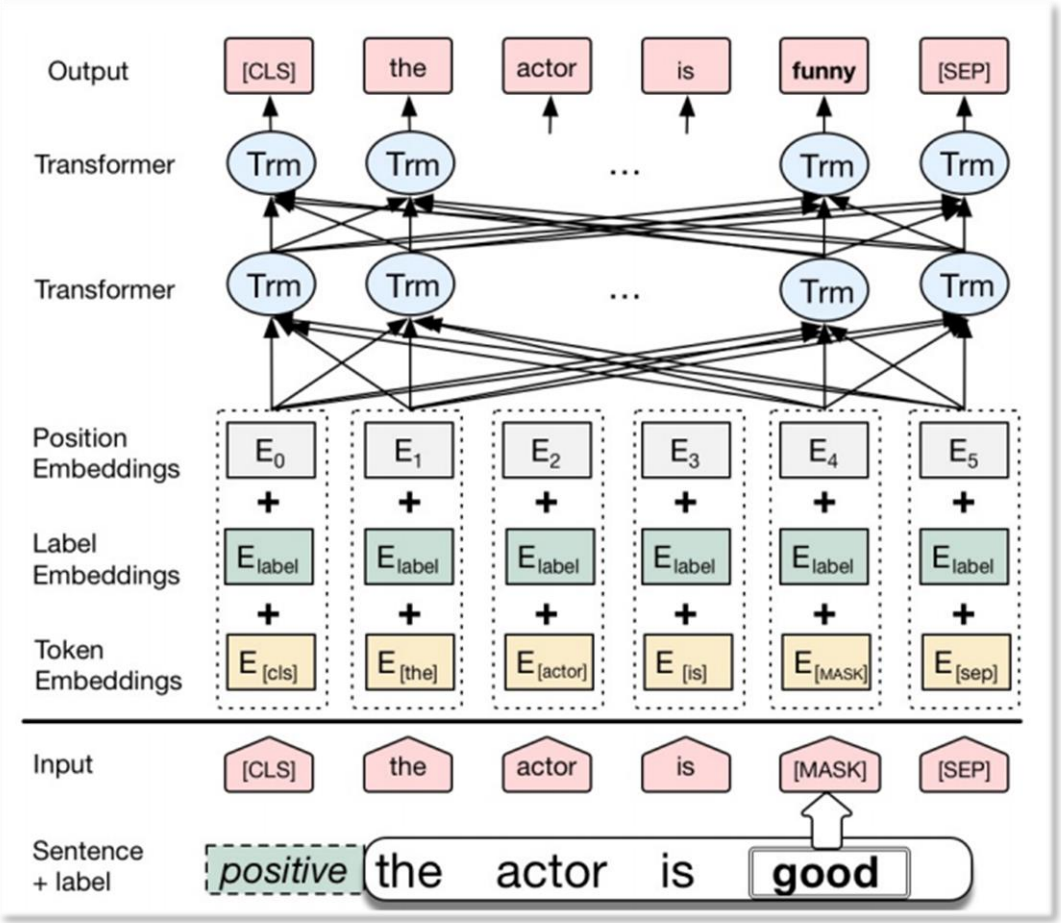
이 값이 1에 수렴하도록 학습

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right)$$

해당 토큰의 target class

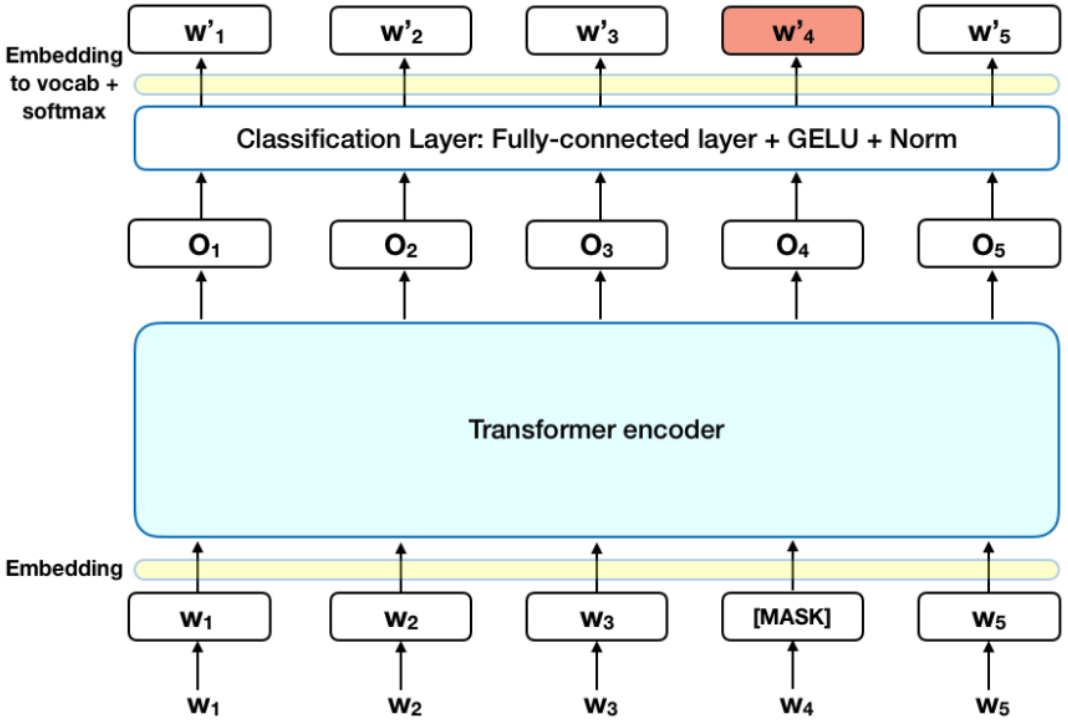
$$\text{loss} = \frac{\sum_{i=1}^{N} loss(i, class[i])}{\sum_{i=1}^{N} weight[class[i]]}$$

모든 토큰에 대한 평균 loss



ELECTRA 제안 배경

- ✓ 최근 Language Model의 pre-training 방법은 많은 연산을 필요로 함
- ✓ MLM은 대부분의 연산이 bidirectional representation에 사용
  - Context를 반영하는데 매우 많은 연산이 사용
  - Input Sequence의 15%만 학습하는 구조로 인해 발생되는 비효율



어떻게 효율을 높일까?

replaced token detection

을 적용하여 학습 효율을 높여보자

1 Introduction

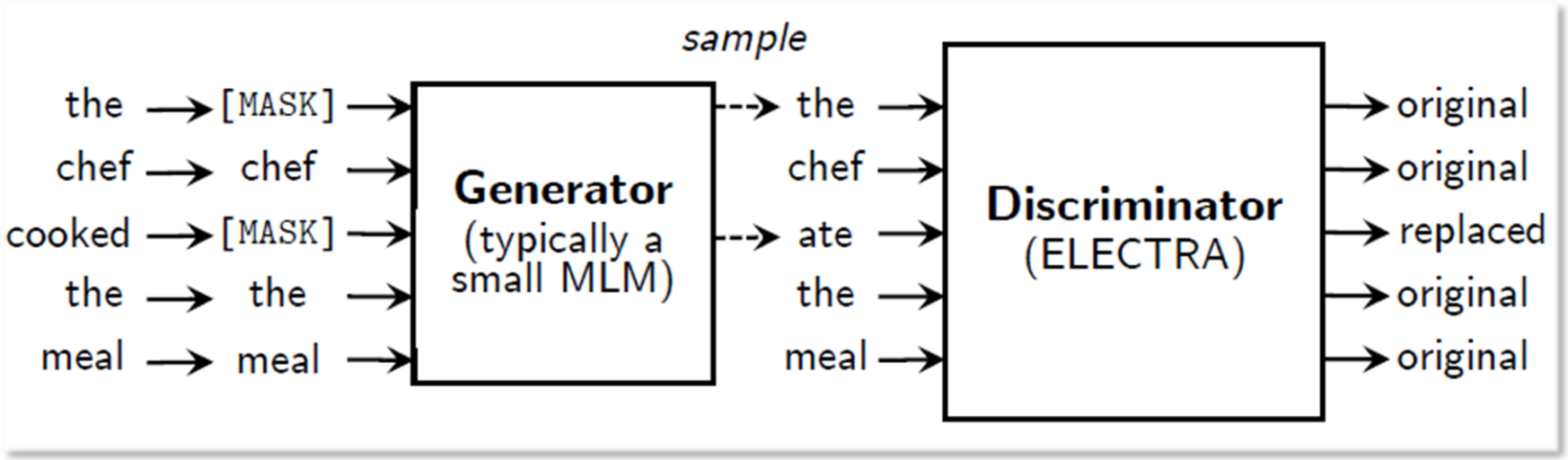
ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately)

- ✓ Generator + Discriminator Network로 구성
- ✓ GAN과 유사한 구조지만 Adversarial 하지는 않음

① Generator는 Input Sequence에 대해 랜덤으로 Masking

② ***Token Replacement*** : Generator는 Masking된 Token을 Replace (Sampling)

③ ***Detection*** : Discriminator는 Sampling된 Sequence에 대해 Binary Classification



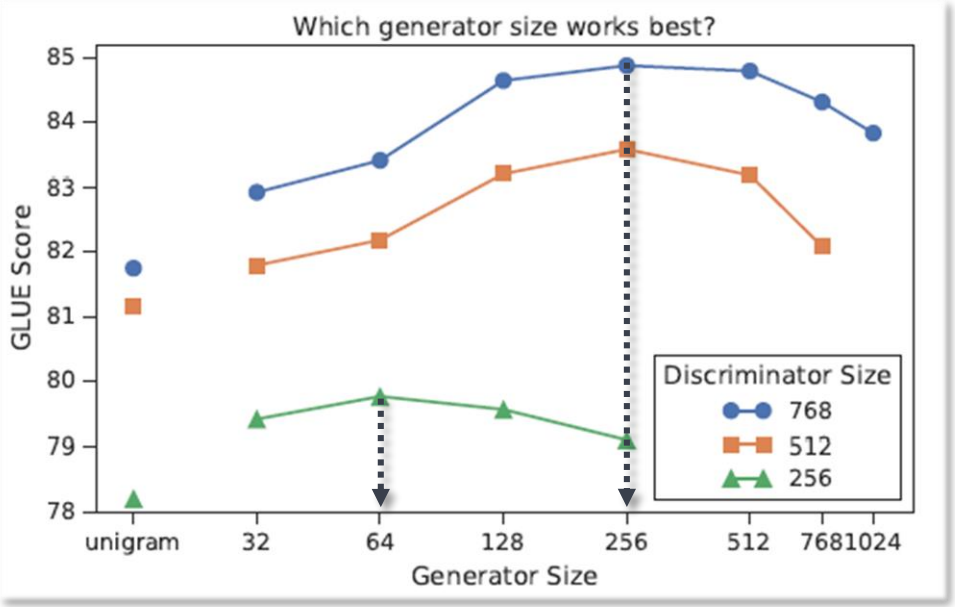
1 Introduction

ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately)

- ✓ Generator는 Discriminator의 ½ ~ ¼ 의 사이즈를 가짐
  - 모델의 크기를 동일하게 할 경우 Weight tying을 통해 약간의 성능향상이 있음
  - 하지만, Network의 사이즈를 동일하게 가져갈 경우 모델의 사이즈가 2배로 커짐
  - 실험적으로 모델 크기 대비 성능비가 가장 좋음
- ✓ 각 Network는 Token Embedding만 공유
- ✓ Combined Loss를 최소화

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

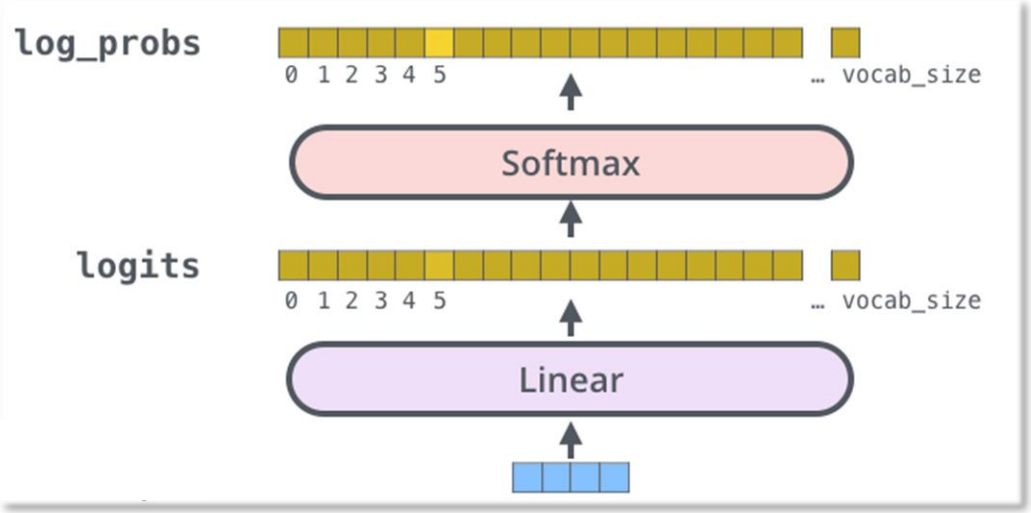
  - λ = 50 으로 설정
    - Discriminator의 Loss가 상대적으로 많이 작음



- ✓ RoBERTa, XLNet의 ¼의 부동 소수점 연산(FLOPs)만으로 동일한 성능

RoBERTa, XLNet보다 4배 효율적으로 학습

Generator



- ✓ 일반적으로 Small MLM을 사용
- ✓ Discriminator를 학습시키기 위한 Network
- ✓ Token Sequence를 입력받아 Contextualized Sequence를 출력
  - Input Sequence : $x = [x_1, \dots, x_n]$
  - Output Sequence : $h(x) = [h_1, \dots, h_n]$
    - $h(x)$의 각 벡터는 해당 위치에서 Context를 반영한 확률분포를 내포함
    - h_t에 Linear Layer + Softmax를 적용하면 확률분포를 얻을 수 있음
- ✓ 주어진 분포에서 가장 확률이 높은 클래스(Token id)로 치환
- ✓ 인풋과 아웃풋의 유사도(내적)를 통해 확률을 정의

Input Sequence의 t번째 Token id

Input Token id를 통해 Word Embedding을 가져옴

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

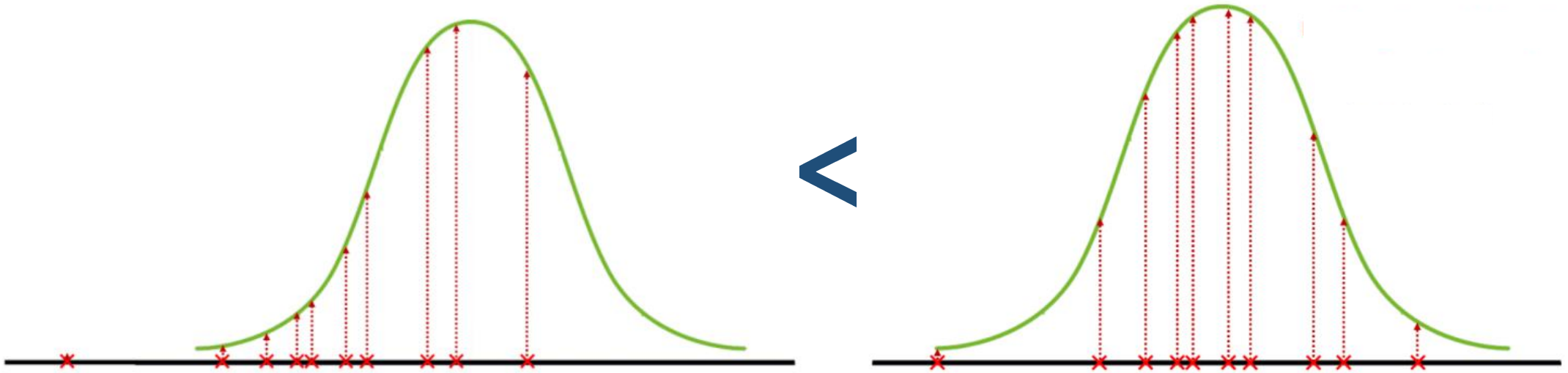
주어진 Input Sequence

Output Contextualized Vector = h_t

Generator : Loss function

✓ Maximum likelihood로 학습

- 주어진 데이터를 가장 잘 설명할 수 있는 확률분포(θ)를 찾는 것
- **Likelihood(가능도)란?** 주어진 확률분포(θ)에 대한 주어진 데이터의 확률밀도함수(p.d.f)값의 곱



우측 분포의 Likelihood가 더 높음

Generator : Loss function

- ✓ $\prod P_G(x_t|X)$를 최대화 하도록 학습 = Negative log likelihood(Loss function)를 최소화 하도록 학습
  - **log를 적용한 이유?** 곱을 합으로 계산하여 언더플로우 방지
  - **Negative Scaling한 이유?** Loss는 최소화 해야하므로
- ✓ Masked position에 대한 Negative log likelihood의 평균이 Loss

모델의 parameters
모델의 확률분포

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

주어진 Input Sequence

전체 토큰의 15%

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

- ✓ CrossEntropyLoss로 구현
  - 내부적으로 log_softmax를 적용
  - Class에 대한 weight 없음

Discriminator

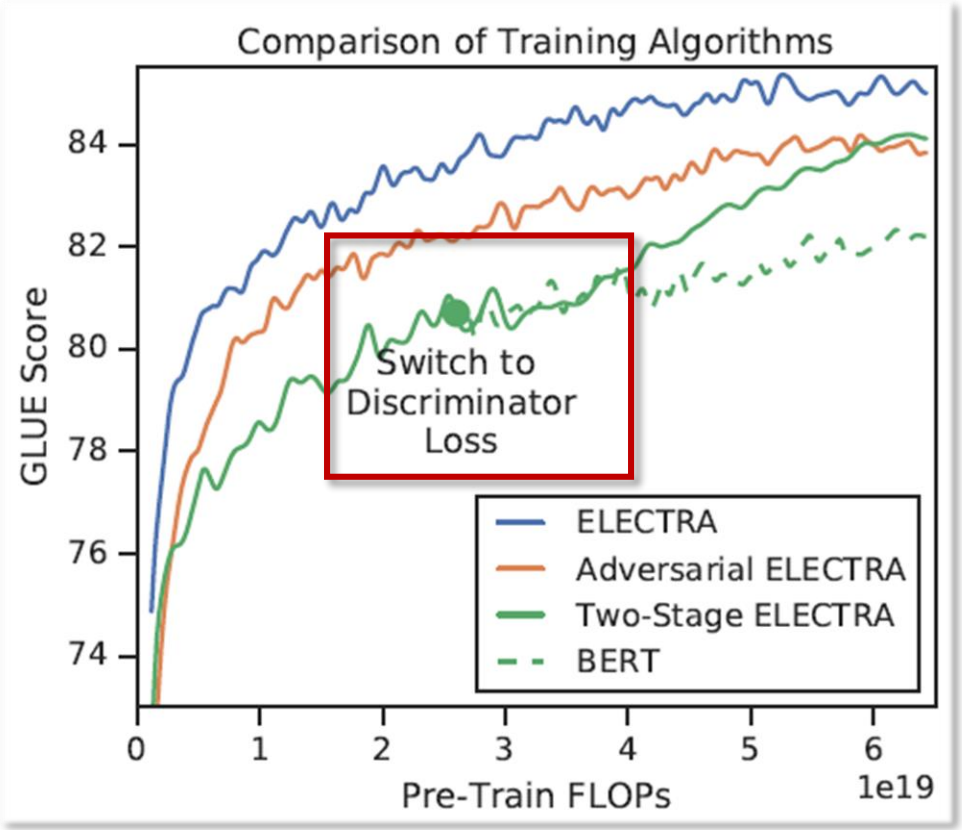
- ✓ Replaced Sequence를 Input으로 받아 모든 토큰에 대해서 Binary Classification
  - Real(0) : Not Replaced
  - Fake(1) : Replaced
- ✓ 모든 토큰에 대해 학습하기 때문에 MLM에 비해 학습 효율이 높음

- ✓ **모든 학습이 끝나면**

  Generator는 버리고

  Discriminator(ELECTRA)만을 이용해

  Down-stream Task에 대해 Fine-tuning 수행



Discriminator : Loss function

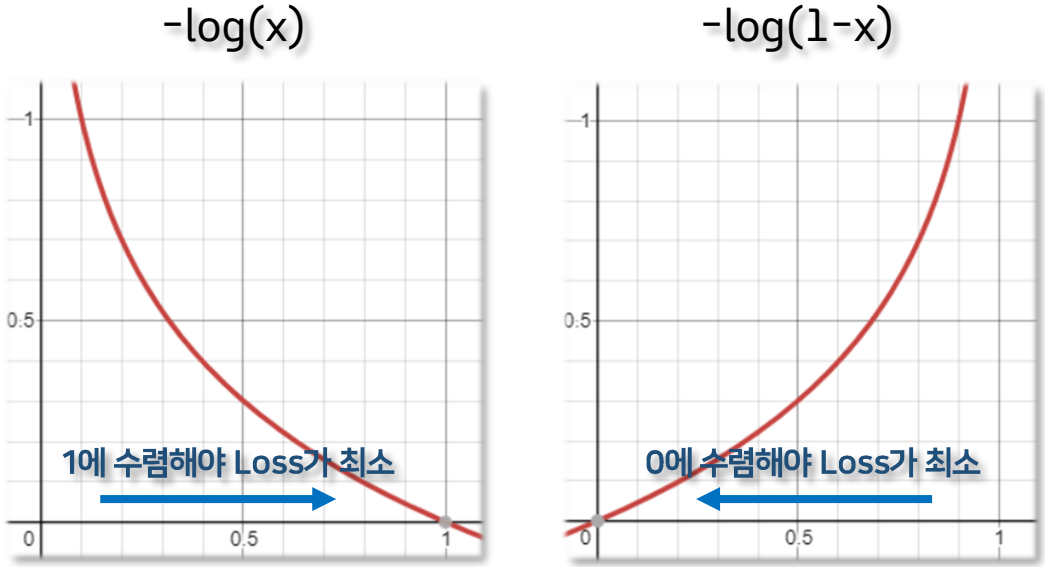
Replaced Sequence

Linear Layer(d_{model} → 1)

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

Position

Fake ~ 1
Real ~ 0



✓ Discriminator가 정답을 맞췄을 경우 Loss가 감소하도록 Loss function을 정의 (Binary Cross Entropy Loss)

- **Input = Output** : D($x^{corrupt}$, t) ≒ 1이면 Loss ≒ 0
- **Input != Output** : D($x^{corrupt}$, t) ≒ 0이면 Loss ≒ 0

원본과 동일할 때

원본과 다를 때

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

→ **모든 토큰에 대해 학습**

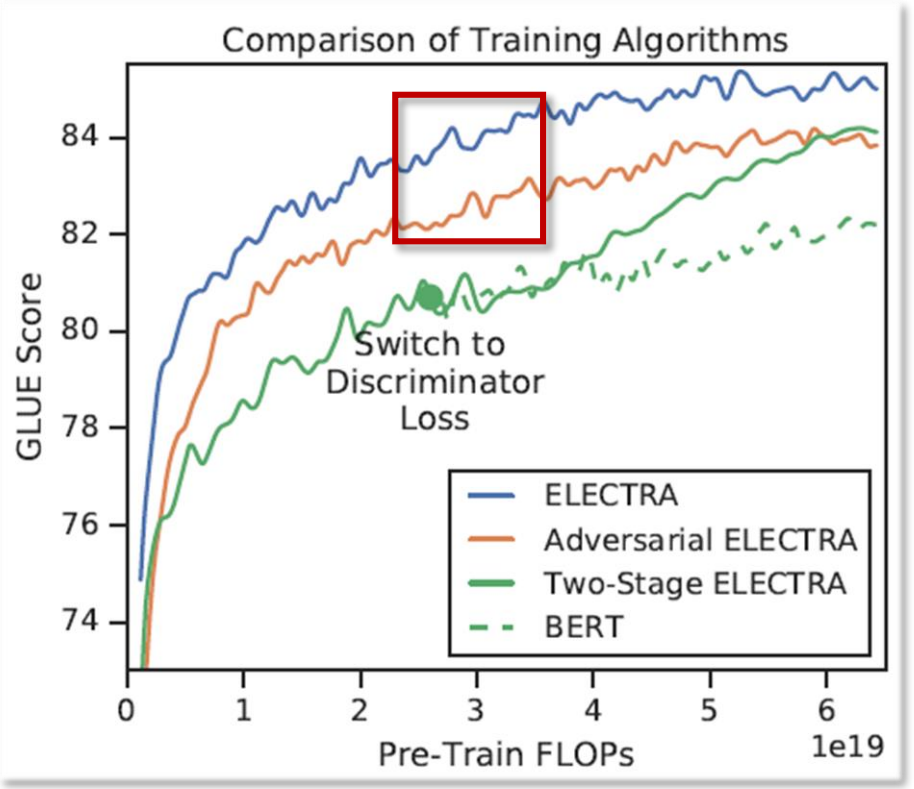
Adversarial training

- ✓ Generator에서 Sampling 때문에 back-propagation 불가능
  - Sampling 하는 행위는 독립적으로 시행되지 않음
  - Sampling 하는 행위는 모든 토큰에 대한 확률의 곱으로 볼 수 있음

- ✓ 몇가지 가정을 통해 복잡한 행위를 단순화 해보자
  - Discriminator의 예측값(=D(x,t))은 다른 토큰에 독립
  - Not Replaced Token들은 Replaced Token들에 독립

- ✓ Reinforcement Learing을 통해서 generator에 대해 학습을 시도
  - policy gradient reinforcement learning

$$\nabla_{\theta_G}\mathcal{L}_{\text{Disc}} \approx \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{m}} \sum_{t\in\boldsymbol{m}} \mathop{\mathbb{E}}_{\hat{x}_t\sim p_G} \nabla_{\theta_g} \log p_G(\hat{x}_t|\boldsymbol{x}^{\text{masked}})[R(\hat{x}_t,\boldsymbol{x}) - b(\boldsymbol{x}^{\text{masked}},t)]$$



➡ MLE Learing보다 성능이 좋지 않음

Small Models

- ✓ ELECTRA의 목적은 Small Model이 Single GPU에서도 빠르게 학습이 가능하도록 하는 것
- ✓ BERT(base, small)와 동일한 parameter setting으로 실험

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

→ ① 동일 연산시 BERT보다 훨씬 좋은 성능

② Small Model에서 ⅛의 학습만으로 BERT능가

Large Models

- ✓ Token Replacement Detection의 효율성을 측정하기 위함
- ✓ $BERT_{LARGE}$와 동일한 parameter setting으로 실험

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**

① RoBERTa, XLNET의 ¼의 학습만으로 동일한 성능

② SOTA Model과 동일 연산 시 더 높은 성능

Large Models

- ✓ Parameter 수 > 3억 인 모델들에 대해서 벤치마크
- ✓ GLUE test-set에 대한 결과
- ✓ ELECTRA(Fully Trained Large) 모델이 가장 좋은 성능을 보임

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**

- ✓ SQuAD 벤치마크에 대해서도 동일한 양상을 보임

Efficiency Analysis

- ✓ ELECTRA는 모든 토큰에 대해서 학습을 하기 때문에 효율적이라는 것은 명백하지는 않음
- ✓ ELECTRA가 어디서 성능상 이득을 얻는지 명확하게 파악하기 위해 실험

- **ELECTRA 15%** : Discriminator가 Masked Token에 대해서만 학습
  - 모든 토큰을 학습함으로써 얻는 이득을 측정하기 위함
- **Replace MLM** : Masking하는 과정없이 Generator가 생성한 토큰으로 치환
  - [MASK]가 모델에 미치는 영향을 측정하기 위함
- **All-Token MLM** : Replace MLM + 모든 토큰에 대해 수행
  - 모든 토큰을 학습함으로써 얻는 이득을 측정하기 위함

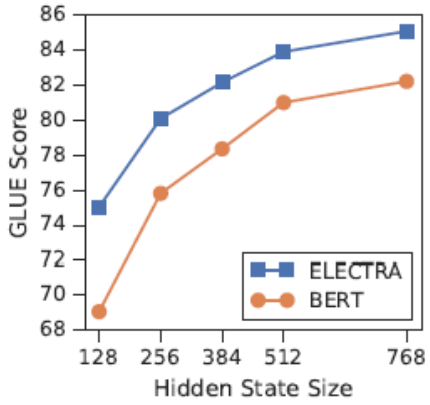
Efficiency Analysis

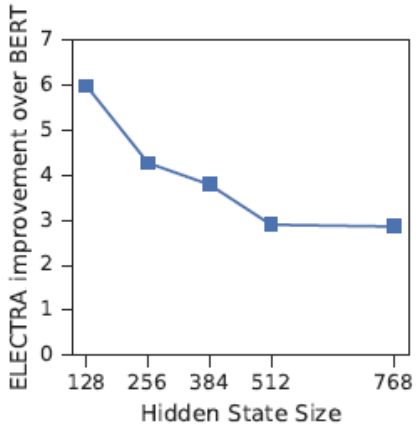
✓ 모든 토큰을 학습하는 과정에서 많은 이득을 얻고 있음을 확인
- ELECTRA 비교
- AT-MLM vs R-MLM

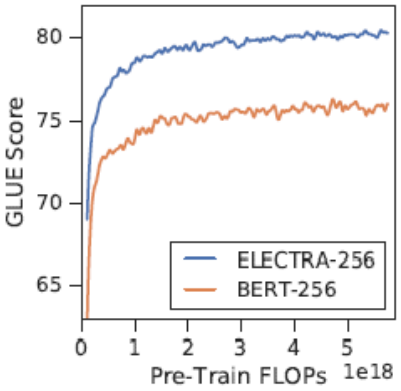
✓ [MASK] 토큰에 대한 Mismatch로 학습하는 과정이 성능을 약간 저하시킴
- BERT vs R-MLM

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).







✓ ELECTRA는 Small 모델에서 더 효율적

Conclusion

- ✓ Token Replaced Detection이라는 새로운 pre-training 방식을 제안
- ✓ Generator에서 생성한 Negative Sample에 대해서 학습
- ✓ MLM에 비해 높은 학습효율을 보임을 증명함
- ✓ Downstream Task들에 더 좋은 성능을 보임
- ✓ **Language Model에 대한 접근성이 크게 향상될 것으로 기대**
- ✓ 앞으로의 pre-training 방법은 **성능뿐만 아니라 학습 효율 또한 고려해야할 것**
  - 모델의 크기가 너무 커졌기 때문

Q / A