ELECTRA

ABSTRACT

INTRODUCTION

Learning Bidirectional Representation이란 BERT를 사용한 학습을 비유한다고 볼 수 있다.
일반적으로 BERT의 Pre-training은 MLM 및 NSP 두 가지의 테스크를 동시에 해결하는 방식을 취한다.
MLM은 토큰의 일정 비율(15%)을 [MASK] 토큰 및 랜덤 토큰(총칭 Noise Token)으로 치환하여 Encoder 모델에 입력 후, Classifier(Linear Layer)를 거쳐 Noise Token을 복호화(Denoising; 다르게 말해 원래 토큰으로 복구)하는 테스크를 수행하는데 이 때문에 Encoder를 **Denoising AutoEncoder** 라고 부른다.
인코더는 입력값에 대해 목표하는 테스크를 수행하기 위한 특징을 추출하기 위한 방식으로 학습해야 하는 목적을 가지고 있다.
인코더는 입력값을 효율적으로 압축하여 테스크에 최적화 되도록 표상(Representation)하여야 하고, 이는 학습을 통해 진행되기 때문에 **Representation Learning** 이라고 부른다.

BRET의 MLM task는 샘플 당 전체 토큰의 15%만을 학습으로 사용하므로 **대용량 데이터셋으로 사전학습을 하는 BERT의 특성상 Substantial Computing Cost(많은 스텝의 학습)가 필요** 하다.

추가 설명
전체 토큰의 15%만을 사용한다는 의미는 최종적으로 복호화된 토큰(Original Token + Noise Token)들 중 Noise Token에 대해서만 선별적으로 Loss를 계산하여 사용하는 것을 의미한다. 이 때문에 한 문장 샘플에 대한 안정적인 학습을 위해서는
해당 문장을 입력으로 하는 스텝을 여러번 거쳐야 한다.

본 논문에서는 이에 대한 대안으로 **Replaced Token Detection** 방식을 제안한다.
이 방식 MLM과 비교하여 모든 토큰에 대해 학습할 수 있다는 장점을 가지고 있다. 한번에 학습에 사용되는 토큰의 비중이 크기 때문에 **Computationally Efficient(계산 효율적)** 이다.
그말인 즉, **적은 자원** 으로도 **빠른 학습** 이 가능하다는 의미.

추가 설명
Replaced Token Detection 테스크는 Masked Token을 복구하는 대신 Synthetically Generated Token(SGT Token)이
진짜 토큰(Real;Not Replaced)인지 가짜 토큰(Fake;Replaced)인지 판별하는 테스크를 수행한다.
SGT Token은 Generator라고 불리는 MLM 모델을 사용하여 생성된다.
이때 MLM 모델은 원론적으로 어떤 것이어도 상관없다.

위와 같은 이유로 이렇게 학습된 모델을 논문에서는 아래와 같이 부른다.

We call our approach ELECTRA for “Efficiently Learning an Encoder that Classifies Token Replacements Accurately.”
여기서 해당 논문이 효율적인 학습을 강조함을 알 수 있다.

앞으로 논문에서 보여줄 내용은 이러한 방식의 학습은(Replaced Token Detection)

BERT 및 여타 SOTA 사전학습 기반 모델들과 비교하여

- 보다 적은 학습 자원을 필요로 함
- 보다 적은 계산효율로 높은 정확도 보임

을 보일 것이다.

METHOD

Replaced Token Detection Task

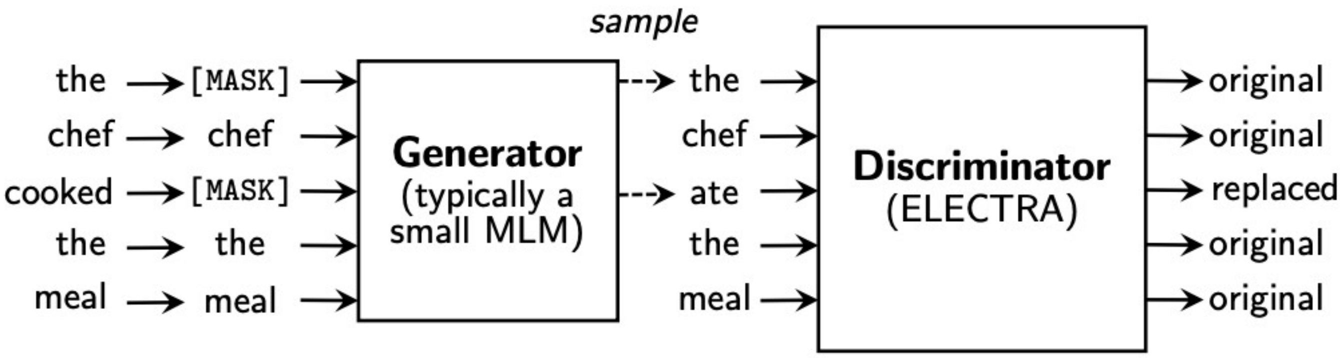


그림 설명
Replaced Token Detection Task는 두가지 모델을 필요로 한다.
각각의 모델은 Generator와 Discriminator로 부르고 각각, Generator는 모든 토큰중 일부[MASK]에 대해 SGT를 생성(MLM Task)하고
Discriminator는 SGT를 포함하는 모든 토큰들에 대해 Fake or Real 판별(Binary Classification Task)을 수행한다.

이러한 학습 과정에서 Generator와 Discriminator는 각각의 토큰들을 최적에 가깝게 표상하는 Representation Vector를 학습하게 된다.

위와 같이 Generator와 Discriminator를 동시에 학습하는 방식은 GAN의 방식과 같다.
다른 점은

1. Generator에 들어가는 값이 임의의 Noise Input이 아니라 토큰의 임베딩 벡터라는 점

   Electra의 Generator는 무작위가 아닌 선별된 샘플을 인풋으로 한다.

2. Generator와 Discriminator가 따로 학습이 된다는 점

   각각이 주어진 정답분포에 가까워지도록(Maximum Likelihood)으로 학습된다.

3. Maximum Likelihood Estimation vs Adversarial Learning

   정말 간결하게 표현하면 이것의 차이는
   Kullback-Leibler Divergence(약칭 KL-Div)와 Jensen-Shannon Divergence(약칭 JSD)의 차이라고 볼 수 있다.

   KL-Div

   KL-Div는 두 분포의 차이를 정량적으로 측정하는 방식이다.
   모델을 통해 나온 확률 분포를 정답 분포와 최대한 가깝게 근사하기 위해 많은 추론 분포와 정답 분포간의 차이를 계산하여 지속적으로 줄여나가는 방식이다.
   이때 모델은 확률을 추정하는 파라미터를 지속적으로 수정/최적화 하게 되며 이러한 과정을 MLE(Maximum Likelihood Estimation)라고 한다.

   좀 더 디테일하게 설명하면 존재 할 수 있는 사건들을 정보량으로 표현 이 정보량의 차이에 대한 평균을 구하는 방식(Entropy)이다.
   즉, E(-log(p_real) + log(q_model)) , where E() is Expectation (i.e KL(P||Q))
   청사진(정답 분포)이 주어지고 이 분포에 근사하는 과정이다.

   JSD

   JSD는 두 분포의 유사성을 대등하게 비교/측정하는 방식이다.
   다르게 말해, 두 분포 중에 어떤 분포가 다른 한 분포의 청사진으로 참조 되는 것이 아닌 두 분포가 서로 대등하게 가까워 지기를 원할 때 사용하는 방식이다.
   두 확률 분포가 존재한다면(예컨데 P,Q), P는 Q를 청사진으로 삼고 Q는 P를 청사진으로 삼는 것이다.
   이는 1/2*KL(P||(P+Q)/2)+1/2*KL(Q||(P+Q)/2)로 표현이 가능하다. (i.e JSD(P||Q) = JSD(Q||P))

Objective Function(Loss Function)

ELECTRA의 최적화 방식은 Generator와 Discriminator의 Cross Entropy를 최소화 하는 것을 목적으로 한다.

보충 설명

Generator는 각각의 [MASK] 토큰이 실제 어떤 토큰이어야 하는지 그 확률 분포를 근사하고,
Discriminator는 각각의 SGT가 Real/Fake인지 그 확률 분포를 근사한다.

Generator와 Discriminator의 학습은 Maximum Likelihood를 추정하는 방식(MLE)이고,
다르게 말해 정답에 가까운 분포를 학습하는 것이다.
앞서 설명 했듯이 정답에 가까운 분포를 만들기 위해서는 KL-Div를 사용한다.
Cross Entropy는 KL-Div의 특수한 형태이다.
정답 분포는 One-Hot.(i.e All or None)일 때, Cross Entropy의 형태가 된다.(더 자세한 내용은 구글 검색하기 바람)

그럼, 수식을 좀 더 자세히 살펴보자

Generator Loss

Generator는 입력 토큰을 받아 해당 자리에 정확한 토큰을 치환할 수 있게끔 Representation Vector를 학습한다.
학습을 위한 목적 함수는 다음과 같은 방식으로 최적화 된다.

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

x, x'은 하나의 input token을 의미한다.
h_G()는 Generator 모델을 의미한다.
e()는 Embedding 레이어를 의미한다.

Generator 모델은 Look-up Table을 제공한다.
그말인 즉, 모든 가능한 토큰(전체 사전)에 대하여 토큰에 대응하는 Embedding Matrix를 만든다.
Embedding 레이어는 입력 토큰에 대해 Embedding Vector로 치환하고
이 값이 Generator를 거쳐 Representation Vector로 변환
다시, Header Network를 빠져나와 Embedding Maxtrix를 생성한다.

h_G(x) , vocab_size x len(embedding vector)
e(x) , 1 x len(embedding vector)

따라서 e(x)*h_G(x)^T , 1 x vocab_size
토큰 각각에 대한 Embedding Vector는 Eembedding Matrix(i.e Look-Up Table)와 곱해지게 되고,
이는 각각의 임베딩 벡터들 간의 유사도 계산과 같다.(유사한 벡터라면 높은 값을, 반대면 낮은 값이 주어진다.)

입력 토큰 각각에 대하여 위와 같은 연산을 수행할 수 있으며 이를 Softmax 취하면, 확률 분포로 표현된다.
여기까지 위의 수식에 대한 설명

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

모델에서 나온 확률 분포가 만들어진다면, 우리는 이 확률 분포로 정답 분포와의 차이 계산할 수 있다.
이 차이는 KL-Div로 계산되며 정답 분포는 One-Hot이기 때문에 KL-Div의 특수한 형태인 Cross Entropy로 계산된다.
여기까지 위의 수식에 대한 설명

Discriminator Loss

Discriminator는 입력 토큰이 Fake/Real인지 분류하는 작업을 한다. 이는 이진분류이다.

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

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

잘 살펴보면 Discriminator의 확률분포 함수와 손실 함수는 Generator에서 사용한 Softmax와 CrossEntropy의 이원화(Binary)버전이다.

We minimize the combined loss

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

최종 목적 함수는 Generator와 Discriminator Loss의 선형 조합으로 이루어진다.
여기서 lambda는 가중치 값으로 Discriminator에 추가되는데,
이 이유에 대해서 해설하자면,
Generator의 Task(다중 분류)가 Discriminator의 Task(이원 분류)보다, 더 어려운 과제이므로 Entropy의 불균형을 야기시킬 수 있다.
Entropy는 불확실 할 수록 높은 값을 가지는데, 직관적으로도 다양한 클래스중 하나를 맞추는 과제가 둘 중 하나의 클래스를 맞추는 과제보다 훨씬 불확실할 가능성이 높을 것이다.
기본적으로 BERT의 Vocab_size는 30000정도 되기 때문에, 30000개 중에 하나의 클래스를 찾아야 하고, 이는 학습시 상당히 높은 entropy를 가질 가능성이 높다.
이 때문에, Discriminator에 일정한 Weight값을 Hyper Parameter로 설정해 주는 것이다.

Method를 정리하자면,

- Replaced Token Detection Task 학습을 채택했다.
- Maximun Likelihood Estimation을 수행한다.
- Discriminator Loss에 가중치를 붙인다.

정도로 정리할 수 있다.

Experiment

Experimental Setup

평가 대상(데이터셋)

General Language Understanding Evaluation(GLUE)와 Stanford Question Answering (SQuAD) dataset를 사용.

해당 데이터셋의 소개는 링크로 남긴다.
GLUE
[[https://huffon.github.io/2019/11/16/glue/]]
SQuAD
[[https://happygrammer.github.io/nlp/dataset/]]

모델 아키텍처

BERT의 아키텍쳐와 하이퍼파라미터를 사용

GLUE 테스크에 대해서는
Simple Linear Classifier를 Header Network로 사용

SQuAD 테스크에 대해서는
XL-NET의 Header Network를 사용
논문에서는 BERT에서 사용한 Header Network(Independently predict)와 다르게 Jointly predict한다고 함.

모델화 방법(Model Extentions)

기본적으로 ELECTRA 모델의 골격은 따로 언급이 없는 이상 BERT-base의 사이즈와 하이퍼파라미터를 따른다.

성능 평가를 위한 모델 비교군 다양화를 위해 다음과 같은 몇가지의 모델 변형을 준다.

1. Weight Sharing
 - Discriminator의 Generator의 Embedding Layer의 가중치를 공유한다.(i.e Embedding Layer Look-Up table 공유)

2. Smaller Generator
 - ELECTRA를 학습할 때, Generator와 Discriminator를 같은 사이즈로 두고 학습할 경우 BERT를 아키텍쳐로 삼을 경우.
   일반 BERT의 사전학습에 두배에 달하는 컴퓨팅 자원을 소모한다.
   이에 대해, Generator의 Layer 사이즈를 몇가지 배율로 줄인다.
 - Generator를 Unigram을(Character Level)에서 생성하는 모델을 학습하는 모델도 만듦
 - 모든 비교 모델에 대해 500k step으로 학습을 진행

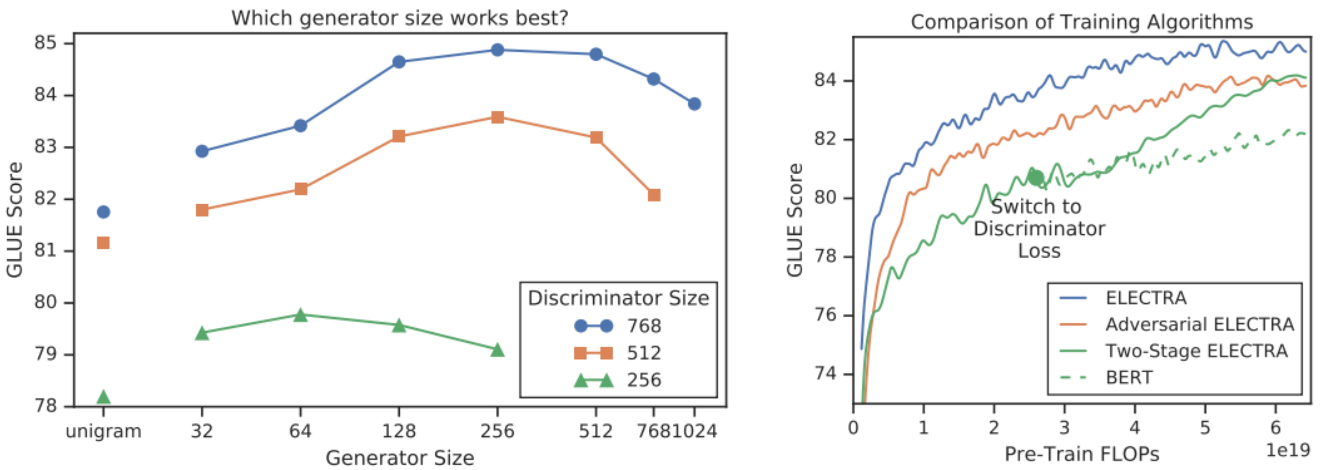
3. Training Algorithm(Two-Step Training)
 - Generator와 Discriminator를 동시에 학습하는 대신 Generator를 미리 n-step만큼 학습한 후,
   학습된 파라미터를 Discriminator에 초기 가중치로 설정하여 n-step학습을 진행(이때, Generator의 가중치는 업데이트 하지 않음)
 - 이 과정을 반복한다.

이하 SMALL MODEL과 LARGE MODEL의 성능을 비교한다. 각각을 구분하여 성능 비교하는 이유는 밑에서 후술

SMALL MODEL

As a goal of this work is to improve the efficiency of pre-training, we develop a small model that can be quickly trained on a single GPU.

SMALL 모델을 비교를 위해 보이고자 하는 것은 학습의 효율성을 다른 모델들과 비교하기 위함이다.



왼쪽 그림은 Generator의 hidden 사이즈를 discriminator hidden 사이즈보다 1/2내지 1/4로 줄일 때, 성능이 가장 좋음을 보여준다.
왼쪽 그림은 ELECTRA가 BERT보다 성능이 좋음을 전반적으로 보여준다.
ELECTRA의 Training Algorithm별 비교에서 일반적인 학습 방식이 적대적 학습(Adversarial Training)과 두 단계 학습(Two-Stage Training)에 비해 성능이 좋음을 보여준다.
여기서 개인적인 의견으로 적대적 학습은 JSD를 목적함수로 사용한 방식을 말한 것 같다.

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

이 표에서 주의깊게 봐야할 포인트는 크게 네가지이다.
1. 파라미터의 수가 많은 모델보다 성능이 좋다.
ELMO와 GPT보다 압도적으로 작은 파라미터의 수로 두 모델을 능가하는 성능을 기록한다.
특히, BERT-small 모델은 두 모델보다 성능이 좋지 않지만, ELECTRA-small은 성능을 능가한다.
2. 적은 스텝의 학습으로 높은 성능을 기록
BERT와 비교해 절반의 학습 스텝으로 이미 BERT의 성능을 추월했다.
3. BERT보다 좋은 성능
BERT-small/base 모두 BERT보다 성능이 좋다.
4. 1 GPU로 학습
1 GPU로 비교 모델보다 높은 성능을 기록 가능

LARGE MODEL

We train big ELECTRA models to measure the effectiveness of the replaced token detection pretraining task at the large scale of current state-of-the-art pre-trained Transformers.

LARGE 모델을 비교하여 보이고자 하는 것은 Replaced Token Detection Task가 다른 SOTA 모델의 학습 방식보다 좋음을 보이기 위함이다.

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**

위의 표에서 주요하게 확인해야 할 포인트는 RoBERTa와 ELECTRA의 비교다.
RoBERTa-500k와 ELECTRA-400k와 비교할 때, 비슷한 파라미터의 개수로 ELECTRA가 1/4배의 학습량으로 RoBERTa의 성능을 따라잡았다.
ELECTRA-1.75M와의 비교에서는 RoBERTa-500k 만큼의 계산량으로 전체 비교 모델을 전반적으로 능가하는 성능을 보여준다.
이를 통해서 ELECTRA의 학습 방식은 다른 모델의 학습 방식에 비해서 빠른 효율적이고 빠른 정확도 상승을 이루기에 적합하다고 볼 수 있다.

EFFICIENCY ANALYSIS

ELECTRA의 학습 방식이 어떠한 요인으로 효율적인지를 파악하기 위한 추가적인 분석을 한다.

세가지의 요인을 확인하는데, 아래와 같다.

1. 전체 토큰의 추론 결과를 학습시 Loss 계산에 사용
2. Masked Language Model Task vs Replaced Token Detection Task
3. [MASK]토큰 사용 유무

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).

결과에 대한 해석은 다음과 같다.
1. 전체 추론 결과를 Loss 계산에 사용하는 것은 성능을 높인다.
2. [MASK] 토큰을 사용하지 않는 것은 성능을 높인다.
3. Replaced Token Detection Task는 Masked Language Model Task에 비해 성능을 높인다

따라서, ELECTRA의 성능 향상 요인은 위의 세가지 요인의 시너지 효과로 볼 수 있다.

파일

Screen Shot 2022-05-01 at 11.49.40 PM.png	55.4 KB	2022-05-01	최한준
Screen Shot 2022-05-02 at 12.41.59 AM.png	15 KB	2022-05-01	최한준
Screen Shot 2022-05-02 at 12.42.36 AM.png	14.9 KB	2022-05-01	최한준
Screen Shot 2022-05-02 at 12.42.36 AM.png	14.9 KB	2022-05-01	최한준
Screen Shot 2022-05-02 at 12.56.44 AM.png	14.9 KB	2022-05-01	최한준
Screen Shot 2022-05-02 at 1.01.26 AM.png	9.81 KB	2022-05-01	최한준
Screen Shot 2022-05-02 at 1.05.57 AM.png	20.2 KB	2022-05-01	최한준
Screen Shot 2022-05-02 at 1.11.42 AM.png	19.9 KB	2022-05-01	최한준
Screen Shot 2022-05-02 at 1.43.02 AM.png	106 KB	2022-05-01	최한준
Screen Shot 2022-05-02 at 1.49.15 AM.png	107 KB	2022-05-01	최한준
Screen Shot 2022-05-02 at 2.04.31 AM.png	115 KB	2022-05-01	최한준
Screen Shot 2022-05-02 at 2.38.35 AM.png	30.6 KB	2022-05-01	최한준