COCO-LM: Correcting and Contrasting Text Sequences for Language Model Pretraining

Author

- 저자: Yu Meng1∗, Chenyan Xiong2, Payal Bajaj2, Saurabh Tiwary2, Paul Bennett2, Jiawei Han1, Xia Song2
  1 University of Illinois at Urbana-Champaign 2 Microsoft
    - NeurIPS 2021 논문

- github: https://github.com/microsoft/COCO-LM

220622 윤주성

Summary

- self-supervised learning framework, COCO-LM: PLM by `COrrecting` and `COntrasting` corrupted text sequences
  - `COrrecting` : ELECTRA 개선 버전
    - 참고: 논문 나온 순서는 대략 ELECTRA - COCO-LM - SimCSE 가 됨
    - **RTD를 copy mechanism에 녹여서 multi-task learning으로 All-token MLM 사용**
  - `Contrasting` : sentence similarity를 PLM self-supervised learning 안에 추가함
    - pos: corrupted sequence & croped sequence (굳이 corrupted를 쓴건 sentence-level alteration 때문인건가 싶음)

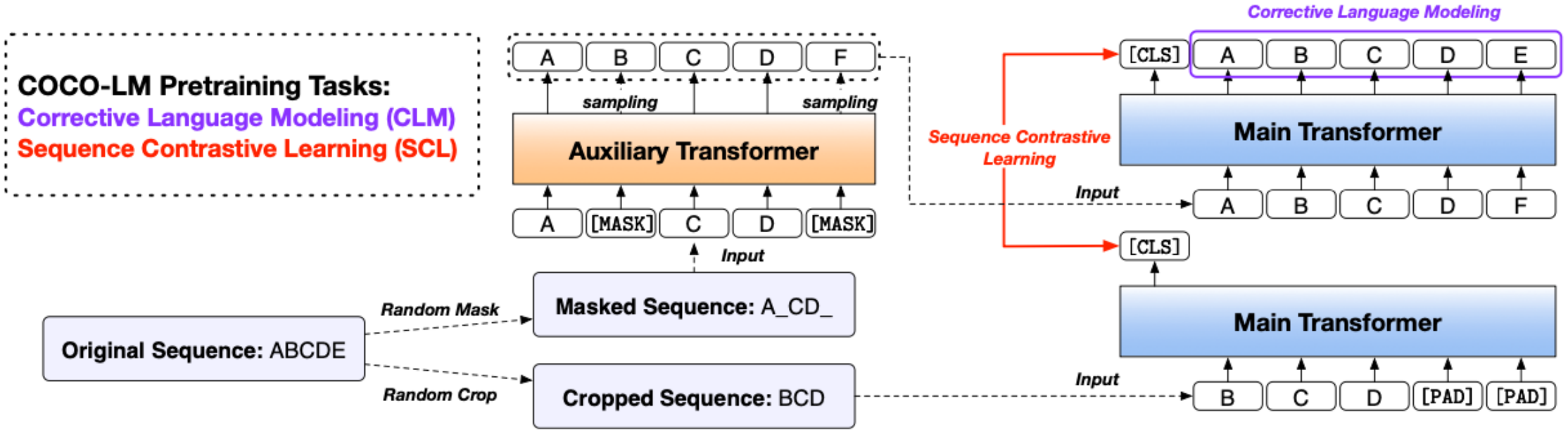


Figure 2: The overview of COCO-LM. The auxiliary Transformer is pretrained by MLM. Its corrupted text sequence is used as the main Transformer's pretraining input in Corrective Language Modeling and paired with the cropped original sequence for Sequence Contrastive Learning.

Related work

- 논문 순서는 대략 ELECTRA - COCO-LM - SimCSE 가 됨
- 약간 ELECTRA 논문의 연장선, 혹은 변형
  - ELECTRA의 contribution -> 15% 계산량 -> 100%로 늘려서 효율 높임
    - input token copy mechanism 실험
  - [MASK] token 대신 generator가 생성한 토큰을 써서 [MASK] 토큰이 일으키는 pre-train/fine-tune discrepancy 제거
  - Replaced Token Detection (RTD) 태스크 제안

Abstract

- `Following ELECTRA-style pretraining`, COCO-LM employs an auxiliary language model to corrupt text sequences, upon which it constructs two new tasks for pretraining the main model
  - The **first token-level task**, Corrective Language Modeling, is to detect and correct tokens replaced by the auxiliary model, in order to better capture token-level semantics. -> **ELECTRA All-Tokens MLM의 개선버전**
  - The s**econd sequence-level task**, Sequence Contrastive Learning, is to align text sequences originated from the same source input while **ensuring uniformity in the representation space**
  - achieves the MNLI accuracy of ELECTRA with `50% of its pretraining GPU hours`. COCO-LM `outperforms the previous best models by 1+ GLUE average points`.

Introduction

- ELECTRA는 pretraining을 효과적이고 효율적으로 개선함
- 하지만 단점도 있음
  - **but pretraining via binary classification hinders the model's usage on applications requiring language modeling capability** (e.g., prompt-based learning [15, 28, 46]). It could further **distort the representation space** as the Transformers are pretrained to output the same "non-replacement" label for all actual tokens.
  - actual token에 대해서는 단순한 output만 출력하기 때문에 representation space에 좋지 않음

Introduction

- ELECTRA-style pretraining 방법에 기반해서 COCO-LM도 auxiliary model(genereator)가 있고 main Transformer로 2가지 token-leval, sequence level PLM task를 하게됨
- The token-level task, **corrective language modeling (CLM)**, pretrains the main Transformer to detect and correct the tokens in the corrupted sequences.
    - It uses a **multi-task setup (copy mechanism(RTD)과 MLM을 동시에 학습)** to combine the benefits of replaced token detection and language modeling.
- The sequence-level task, **sequence contrastive learning (SCL)**, pretrains the model to align text sequences originated from the same source sequence and enforce uniformity of the representation space
- GLUE [54] and SQuAD [41] benchmarks, COCO-LM not only **outperforms state-of-the-art** pretraining approaches in effectiveness, but also significantly improves the pretraining efficiency

Method

- present the preliminaries of PLMs, their challenges, and the new COCO-LM framework.

Preliminary on Language Model Pretraining

BERT Pretraining uses the masked language modeling task (MLM) [11], which is to take an input sequence $X^{\text{orig}} = [x_1^{\text{orig}}, \ldots, x_i^{\text{orig}}, \ldots, x_n^{\text{orig}}]$, with 15% random tokens replaced by `[MASK]` symbols (*e.g.*, the i-th token), and train the model to predict the original tokens at the masked positions:

$$\left[x_1^{\text{orig}}, \ldots, \texttt{[MASK]}_i, \ldots, x_n^{\text{orig}}\right] \xrightarrow{\text{Transformer}} \boldsymbol{H} \xrightarrow{\text{MLM Head}} p_{\text{MLM}}(x|\boldsymbol{h}_i),$$

where the Transformer generates contextualized representations $\boldsymbol{H} = \{\boldsymbol{h}_i\}_{i=1}^n$. The MLM Head predicts the masked token from the vocabulary V using the hidden representation $\boldsymbol{h}_i$ and token embeddings $\boldsymbol{x}$. The pretraining minimizes the MLM loss on the set of masked positions $\mathcal{M}$. Specifically,

$$p_{\text{MLM}}(x|\boldsymbol{h}_i) = \frac{\exp(\boldsymbol{x}^\top \boldsymbol{h}_i)}{\sum_{x_t \in V} \exp(\boldsymbol{x}_t^\top \boldsymbol{h}_i)}; \quad \mathcal{L}_{\text{MLM}} = \mathbb{E}\left(-\sum_{i \in \mathcal{M}} \log p_{\text{MLM}}\left(x_i^{\text{orig}}|\boldsymbol{h}_i\right)\right).$$

Preliminary on Language Model Pretraining

ELECTRA Pretraining uses two Transformers, a "generator" pretrained by MLM, and a "discriminator" pretrained using the generator's outputs. We refer them as *auxiliary* and *main* Transformers, as the former is discarded after pretraining and the latter may be trained by "generative" tasks too.

The auxiliary model outputs a corrupted sequence X^{MLM} by sampling from its predicted probability:

$$x_i^{\text{MLM}} \sim p_{\text{MLM}}\left(x|\boldsymbol{h}_i\right), \text{ if } i \in \mathcal{M}\,; \quad x_i^{\text{MLM}} = x_i^{\text{orig}}, \text{ else.} \tag{1}$$

The masked positions are replaced by sampled tokens considered plausible in context by the auxiliary Transformer, which are more deceiving than random replacements. ELECTRA uses a skinnier auxiliary network (*e.g.*, hidden dimension is $1/3$ of the main model) to control the signal difficulty.

The main Transformer takes X^{MLM} and classifies the replaced tokens:

$$X^{\text{MLM}} \xrightarrow{\text{Main Transformer}} \boldsymbol{H} \xrightarrow{\text{RTD Head}} p_{\text{RTD}}\left(\mathbf{1}(x_i^{\text{MLM}} = x_i^{\text{orig}})\middle|\boldsymbol{h}_i\right),$$

where $\mathbf{1}(\cdot)$ is the indicator function. The Replaced Token Detection (RTD) head uses a sigmoid linear layer to output the binary probability, and the main Transformer is trained with binary cross entropy loss. The RTD task is trained on all tokens instead of masked ones and improves efficiency.

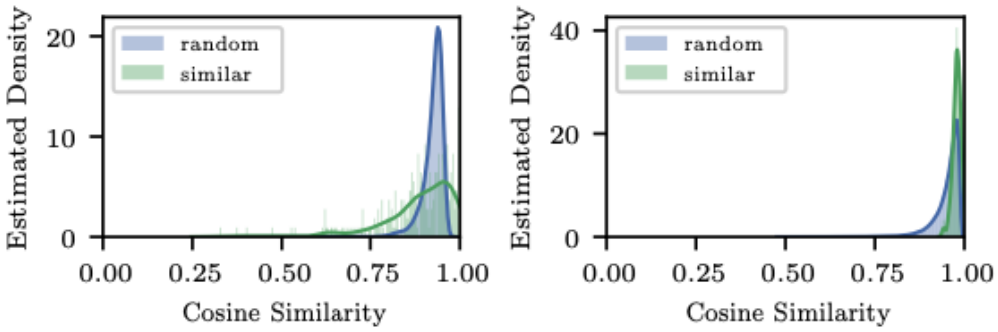
The two Transformers are pretrained jointly. The auxiliary model gradually generates more realistic replacement tokens and the main model learns to better detect them. This forms a natural learning curriculum and significantly improves ELECTRA's accuracy in downstream tasks [7].

Challenges of ELECTRA-Style Pretraining

- Missing Language Modeling Benefits.
    - classification task in ELECTRA is simpler and more stable [61], but raises two challenges.
        - first is the **lack of language modeling capability** which is a necessity in some tasks [6]. For example, prompt-based learning requires a language model to generate labels
            - RTD로 학습해서 LM 역할 잘 못함
        - second is that the binary classification task **may not be sufficient to capture certain word-level semantics** that are critical for token-level tasks

Challenges of ELECTRA-Style Pretraining

- Squeezing Representation Space
  - 기존 transformers도 cos 값이 잘 안나옴
  - 데이터
    - random sentence pairs (from pretraining corpus)
    - semantically similar pairs (from STS-B)
  - the representations from Transformer-based language models often reside in a narrow cone, where `two random sentences have high similarity scores` (`lack of uniformity`)
  - `closely related sentences may have more different representations` (`lack of alignment`)

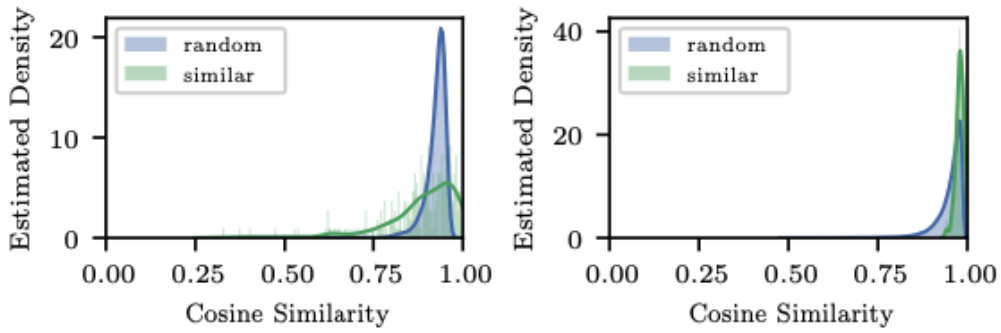


(a) RoBERTa. (b) ELECTRA.

Figure 1: Cosine similarity distributions of random/similar sequence pairs using `[CLS]` embeddings from pretrained models. Histograms/curves are distribution bins/kernel density estimates.

Challenges of ELECTRA-Style Pretraining

- Squeezing Representation Space
  - With RoBERTa, the cosine similarities of most random sentence pairs are near 0.8, bigger than many semantically similar pairs. **The representation space from ELECTRA is even more squeezed**. Nearly all sentence pairs, `both random and similar ones, have around 0.9 cosine similarity`.

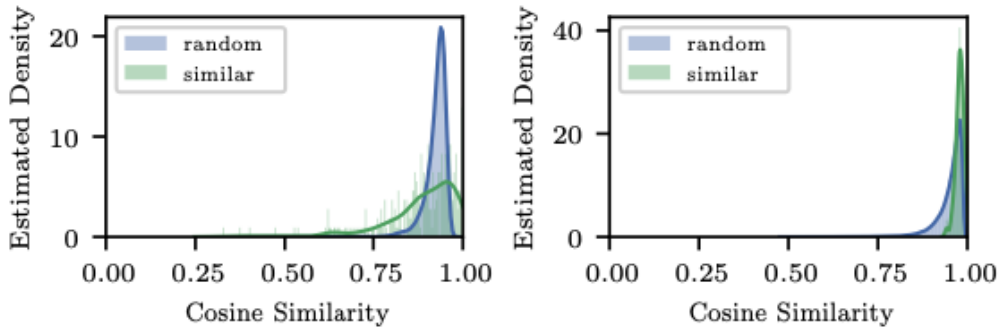


(a) RoBERTa. (b) ELECTRA.

Figure 1: Cosine similarity distributions of random/similar sequence pairs using `[CLS]` embeddings from pretrained models. Histograms/curves are distribution bins/kernel density estimates.

Challenges of ELECTRA-Style Pretraining

- This may not be surprising as **ELECTRA is pretrained to predict the same output** ("non-replacement") for all tokens in these sequences. The **irregular representation space raises the risk of degeneration** [37, 55] and often necessitates sophisticated post-adjustment or fine-tuning to improve the sequence representations [16, 30, 32, 60].



(a) RoBERTa. (b) ELECTRA.

Figure 1: Cosine similarity distributions of random/similar sequence pairs using `[CLS]` embeddings from pretrained models. Histograms/curves are distribution bins/kernel density estimates.

COCO-LM Pretraining

- The auxiliary Transformer is pretrained by masked language modeling (MLM) and generates corrupted sequences.
- The main Transformer is pretrained to correct the corruption (CLM) and to contrast the corrupted sequences with the cropped sequences (SCL)

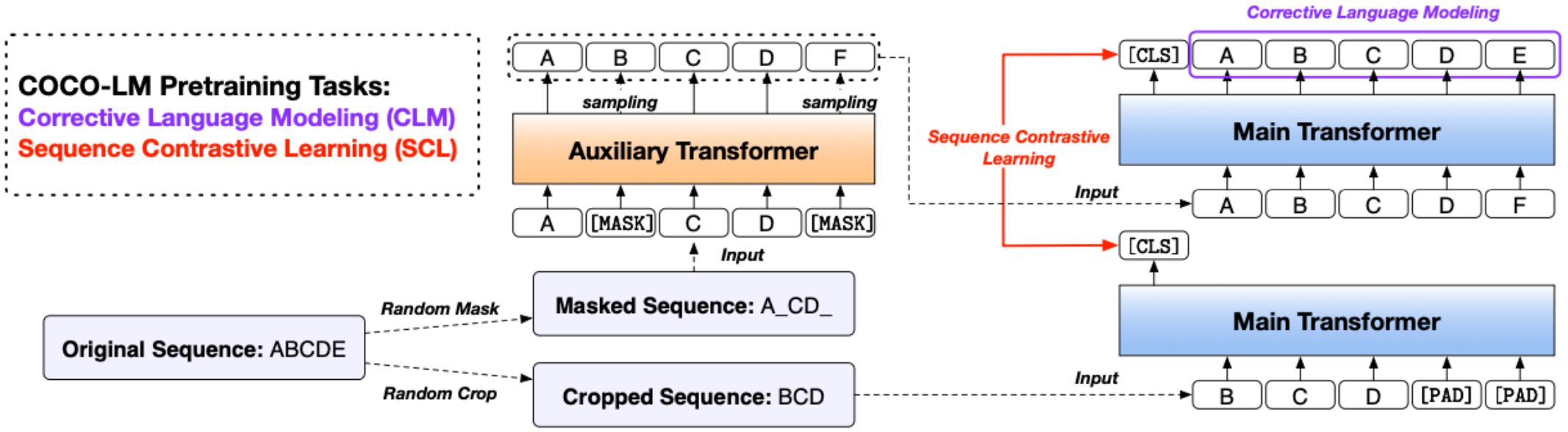


Figure 2: The overview of COCO-LM. The auxiliary Transformer is pretrained by MLM. Its corrupted text sequence is used as the main Transformer's pretraining input in Corrective Language Modeling and paired with the cropped original sequence for Sequence Contrastive Learning.

COCO-LM Pretraining

- X^MLM은 corrupted sequence (aux transformers 결과)
- MLM 확률에 copy 확률 추가
  - ELECTRA All-Token MLM
  - 입력토큰확률에 copy 확률을 더해 advantage를 준다고 보면됨
- corrupted text에 대한 LM은 어렵다
  - ELECTRA 보다 성능 낮음
  - All-token MLM 보완을 위해 multi-task setup 사용

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).

3.3 COCO-LM Pretraining

COCO-LM also employs an auxiliary Transformer to construct the corrupted text sequence, as in Eqn. (1), but it introduces two new pretraining tasks upon the corrupted sequences to address the challenges previously described. In the rest of this section, we present these two tasks and then the detailed configurations of COCO-LM. Its framework is illustrated in Figure 2.

Corrective Language Modeling (CLM) trains the main Transformer to recover the original tokens, given the corrupted text sequence X^{MLM}:

$$X^{\text{MLM}} \xrightarrow{\text{Main Transformer}} \boldsymbol{H} \xrightarrow{\text{CLM Head}} p_{\text{CLM}}(x|\boldsymbol{h}_i).$$

The CLM Head uses the hidden representations $\boldsymbol{H}$ to output a language modeling probability, instead of a binary classification score. The forward pass of the CLM Head is the same as All-Token MLM, a variation of ELECTRA [7] that consists of a language modeling layer and a binary classification layer for the copy mechanism:

$$p_{\text{LM}}(x_i|\boldsymbol{h}_i) = \mathbb{1}\left(x_i = x_i^{\text{MLM}}\right) p_{\text{copy}}(1|\boldsymbol{h}_i) + p_{\text{copy}}(0|\boldsymbol{h}_i) \frac{\exp(\boldsymbol{x}_i^\top \boldsymbol{h}_i)}{\sum_{x_t \in V} \exp(\boldsymbol{x}_t^\top \boldsymbol{h}_i)},$$

$$p_{\text{copy}}(y_i|\boldsymbol{h}_i) = \exp(y_i \cdot \boldsymbol{w}_{\text{copy}}^\top \boldsymbol{h}_i) / \left(\exp(\boldsymbol{w}_{\text{copy}}^\top \boldsymbol{h}_i) + 1\right),$$

where $\boldsymbol{w}_{\text{copy}}$ is a learnable weight and $p_{\text{copy}}(y_i|\boldsymbol{h}_i)$ is the copy mechanism ($y_i = 1$ when the input token is original and can be directly copied to the output; $y_i = 0$ when the input token needs to be corrected to another token from the vocabulary).

In ELECTRA, All-Token MLM performs worse than RTD [7]. Language modeling on the corrupted text sequence X^{MLM} is hard as the replaced tokens from the auxiliary model are more deceiving than `[MASK]`. To improve the language model learning, different from All-Token MLM, CLM employs a

COCO-LM Pretraining

- RTD task를 copy mechanism에 적용
  - `stop gradient` 사용
  - ELECTRA에서도 sigmoid지만, RTD처럼 따로 loss 주는 내용은 없음
- copy mechanism(RTD)와 LM을 동시에 학습 (multi-task learning)
  - LM은 Masking되서 corrupted token이 있는 곳만

multi-task setup that combines the RTD task to explicitly train the copy mechanism $p_{\text{copy}}(\cdot)$:

$$\mathcal{L}_{\text{copy}} = -\mathbb{E}\left(\sum_{i=1}^{n} \mathbb{1}\left(x_i^{\text{MLM}} = x_i^{\text{orig}}\right) \log p_{\text{copy}}(1|\boldsymbol{h}_i) + \mathbb{1}\left(x_i^{\text{MLM}} \neq x_i^{\text{orig}}\right) \log p_{\text{copy}}(0|\boldsymbol{h}_i)\right), \quad (2)$$

$$\mathcal{L}_{\text{LM}} = -\mathbb{E}\left(\sum_{i\in\mathcal{M}} \log p_{\text{LM}}\left(x_i^{\text{orig}}|\boldsymbol{h}_i\right)\right)$$

$$= -\mathbb{E}\left(\sum_{i\in\mathcal{M}} \log\left(\mathbb{1}\left(x_i^{\text{MLM}} = x_i^{\text{orig}}\right) p_{\text{copy}}^{\text{sg}}(1|\boldsymbol{h}_i) + p_{\text{copy}}^{\text{sg}}(0|\boldsymbol{h}_i)\frac{\exp(\boldsymbol{x}_i^\top \boldsymbol{h}_i)}{\sum_{x_t\in V}\exp(\boldsymbol{x}_t^\top \boldsymbol{h}_i)}\right)\right),$$

$$\mathcal{L}_{\text{CLM}} = \lambda_{\text{copy}}\mathcal{L}_{\text{copy}} + \mathcal{L}_{\text{LM}}.$$

The hyperparameter λ_{copy} balances the weights of the two tasks. The binary cross entropy loss in Eqn. (2) explicitly trains the copy probability. We also use stop gradient (sg) to decouple the gradient backpropagation to $p_{\text{copy}}(\cdot)$ from the LM task. This way, the main Transformer first learns the easier classification task and then uses it to help learn the harder LM task. The binary classification task is trained on all tokens while the language modeling task is trained only on masked positions.

CLM combines the advantages of MLM and ELECTRA: The main Transformer is trained on all tokens with the help of the binary classification task while also being able to predict words, thus enjoying the efficiency benefits of ELECTRA and preserving the language modeling benefits.

COCO-LM Pretraining

- sequence-level에서는 문장의 90%정도 crop해서 data augmentation
  - postive set 구축
  - simCSE에서는 dropout으로 했었음
- negative는 배치내에서 남는 sequence 사용

Sequence Contrastive Learning (SCL) forms a contrastive learning objective upon the sequence embeddings to learn more robust representations. Broadly, contrastive learning is to align a positive pair of instances, often different views of the same information [4, 34], in contrast to unrelated negative instances [22, 60]. The different views are often obtained by applying data augmentations on the same input, for example, rotation, cropping, and blurring on visual representations [4, 34], so that the neural networks can learn representations robust to these data alterations.

In COCO-LM, the corrupted sequence X^{MLM} already provides a form of data augmentation. We pair it with another augmentation, X^{crop}, a randomly cropped contiguous span of X^{orig} (the length of X^{crop} is 90% of X^{orig} so that the major sequence meaning is preserved), to construct the positive pair and to contrast with random negatives.

Specifically, a training batch B in SCL includes a random set of corrupted and cropped sequences: $B = \{(X_1^{\text{MLM}}, X_1^{\text{crop}}), \dots, (X_N^{\text{MLM}}, X_N^{\text{crop}})\}$, with X_k^{MLM} and X_k^{crop} originated from X_k^{orig}. A positive contrastive pair (X, X^+) consists of either $(X_k^{\text{MLM}}, X_k^{\text{crop}})$ or $(X_k^{\text{crop}}, X_k^{\text{MLM}})$ (symmetrical contrast). The negative instances are all the remaining sequences in the batch $B^- = B \setminus \{(X, X^+)\}$. The contrastive loss is formulated as:

$$\begin{aligned}\mathcal{L}_{\text{SCL}} &= -\mathbb{E}\left(\log \frac{\exp(\cos(\boldsymbol{s}, \boldsymbol{s}^+)/\tau)}{\exp(\cos(\boldsymbol{s}, \boldsymbol{s}^+)/\tau) + \sum_{X^- \in B^-} \exp(\cos(\boldsymbol{s}, \boldsymbol{s}^-)/\tau)}\right), \\ &= -\mathbb{E}\left(\cos(\boldsymbol{s}, \boldsymbol{s}^+)/\tau - \log\left(\exp(\cos(\boldsymbol{s}, \boldsymbol{s}^+)/\tau) + \sum_{X^- \in B^-} \exp\left(\cos(\boldsymbol{s}, \boldsymbol{s}^-)/\tau\right)\right)\right), \quad (3)\end{aligned}$$

where $\boldsymbol{s}, \boldsymbol{s}^+, \boldsymbol{s}^-$ are the representations of X, X^+, X^-, respectively, from the main Transformer (*i.e.*, $\boldsymbol{h}_{\texttt{[CLS]}}$). The similarity metric is cosine similarity (cos) and the temperature τ is set to 1.

As shown in Wang et al. [55], the first term in Eqn. (3) ($\cos(\boldsymbol{s}, \boldsymbol{s}^+)$) improves *alignment* of the space. It encourages representations to be robust to the corruptions and the alterations on the original text. The second term in Eqn. (3) promotes *uniformity*. It pushes unrelated sequences apart in the representation space and ensures low cosine similarity between random data points. Several studies have observed improved generalization ability from better alignment and uniformity [16, 37, 55].

Aligning X^{MLM} with X^{crop} requires the main Transformer to produce sequence representations robust to both token-level (*i.e.*, MLM replacements) and sequence-level (*i.e.*, cropping) alterations. The model is thus encouraged to reason more using partially altered sequences to recover the original information.

Overall Training. COCO-LM uses the following loss function:

$$\mathcal{L}_{\text{COCO-LM}} = \mathcal{L}_{\text{MLM}}^{\text{Aux.}} + \mathcal{L}_{\text{CLM}}^{\text{Main}} + \mathcal{L}_{\text{SCL}}^{\text{Main}}. \quad (4)$$

COCO-LM Pretraining

- sequence-level에서는 90%정도 crop해서 data augmentation
  - postive set 구축
  - simCSE에서는 dropout으로 했었음
- negative는 배치내에서 남는 sequence 사용

```
def forward(self, src_tokens, span_tokens=None, features_only=False,
            return_all_hiddens=False, classification_head_name=None,
            masked_tokens=None, targets=None, **kwargs):
            # 생략
        if seq_contrast and self.args.span > 0:
            assert span_tokens is not None
            span_padding_mask = get_padding_mask(span_tokens)
            _, extra = self.encoder(
                span_tokens,
                features_only=True,
                return_all_hiddens=return_all_hiddens,
                padding_mask=span_padding_mask,
                seq_contrast=seq_contrast,
                **kwargs
            )
            span_seq_emb = extra["seq_emb"]

    gen_x, extra = self.encoder(
        src_tokens,
        features_only=features_only,
        return_all_hiddens=return_all_hiddens,
        padding_mask=padding_mask,
        masked_tokens=masked_tokens,
        seq_contrast=seq_contrast,
        **kwargs
    )

    if span_seq_emb is not None:
        extra["span_seq_emb"] = span_seq_emb
```

Network Configurations

- auxiliary model
  - Similar to ELECTRA, the auxiliary Transformer is smaller than the main model
  - We reduce the number of layers to **1/3 or 1/4** (under base or large model setup, respectively) but keep its hidden dimension the same with the main model, instead of shrinking its hidden dimensions
  - We disable dropout in it when sampling replacement tokens.
- main model
  - standard architecture of **BERT/ELECTRA**

Experimental Setup

- Pretraining Settings
  - base, base++, and large++. Base is the BERTBase training configuration [11]: Pretraining on Wikipedia and BookCorpus [63] (16 GB of texts) for 256 million samples on 512 token sequences
  - 32, 768 uncased BPE vocabulary
- Model Architecture
  - base/base++ model uses the BERT Base architecture [11]: 12 layer Transformer, 768 hidden size, plus **T5 relative position encoding**.
  - large++ model is the same with BERTLarge, 24 layer and 1024 hidden size, plus T5 relative position encoding
  - auxiliary network uses the same hidden size but a shallow **4-layer Transformer** in base/base++ and a **6-layer** one in large++. When generating XMLM we disable dropout in the auxiliary model

Experimental Setup

- Downstream Tasks
  - GLUE [54] and SQuAD 2.0
  - Standard hyperparameter search in fine-tuning is performed, and the search space can be found in Appendix B.
  - **reported results are the median of five random seeds** on GLUE and SQuAD

Evaluation Results

- COCO-LM outperforms all recent state-of-the-art pretraining models on GLUE average and SQuAD
- MRPC와 STS-B에서도 차이가 나는 모습

Model	Params	GLUE Single Task									SQuAD 2.0	
		MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	RTE	MRPC	STS-B	AVG	EM	F1
Base Setting: BERT Base Size, Wikipedia + Book Corpus (16GB)												
BERT [11]	110M	84.5/-	91.3	91.7	93.2	58.9	68.6	87.3	89.5	83.1	73.7	76.3
RoBERTa [31]	125M	84.7/-	–	–	92.7	–	–	–	–	–	–	79.7
XLNet [62]	110M	85.8/85.4	–	–	92.7	–	–	–	–	–	78.5	81.3
ELECTRA [7]	110M	86.0/85.3	90.0	91.9	93.4	64.3	70.8	84.9	89.1	83.7	80.5	83.3
MC-BERT [61]	110M	85.7/85.2	89.7	91.3	92.3	62.1	75.0	86.0	88.0	83.7	–	–
DeBERTa [23]	134M	86.3/86.2	–	–	–	–	–	–	–	–	79.3	82.5
TUPE [26]	110M	86.2/86.2	91.3	92.2	93.3	63.6	73.6	89.9	89.2	84.9	–	–
RoBERTa (Ours)	110M	85.8/85.5	91.3	92.0	**93.7**	60.1	68.2	87.3	88.5	83.3	77.7	80.5
ELECTRA (Ours)	110M	86.9/86.7	91.9	92.6	93.6	**66.2**	75.1	88.2	89.7	85.5	79.7	82.6
COCO-LM	110M	**88.5/88.3**	**92.0**	**93.1**	93.2	63.9	**84.8**	**91.4**	**90.3**	**87.2**	**82.4**	**85.2**
Base++ Setting: BERT Base Size, Bigger Training Data, and/or More Training Steps												
XLNet [62]	110M	86.8/-	91.4	91.7	94.7	60.2	74.0	88.2	89.5	84.6	80.2	–
RoBERTa [31]	125M	87.6/-	91.9	92.8	94.8	63.6	78.7	90.2	91.2	86.4	80.5	83.7
UniLM V2 [1]	110M	88.5/-	91.7	93.5	**95.1**	65.2	81.3	**91.8**	91.0	87.1	83.3	86.1
DeBERTa [23]	134M	88.8/88.5	–	–	–	–	–	–	–	–	83.1	86.2
CLEAR [59]	110M	86.7/-	90.0	92.9	94.5	64.3	78.3	89.2	89.8	85.7	–	–
COCO-LM	134M	**90.2/90.0**	**92.2**	**94.2**	94.6	**67.3**	**87.4**	91.2	**91.8**	**88.6**	**85.4**	**88.1**
Large++ Setting: BERT Large Size, Bigger Training Data, and More Training Steps												
XLNet [62]	360M	90.8/90.8	92.3	94.9	**97.0**	69.0	85.9	90.8	92.5	89.2	87.9	90.6
RoBERTa [31]	356M	90.2/90.2	92.2	94.7	96.4	68.0	86.6	90.9	92.4	88.9	86.5	89.4
ELECTRA [7]	335M	90.9/-	92.4	95.0	96.9	69.1	88.0	90.8	92.6	89.4	88.0	90.6
DeBERTa [23]	384M	91.1/91.1	92.3	95.3	96.8	70.5	–	–	–	–	88.0	90.7
COCO-LM	367M	**91.4/91.6**	**92.8**	**95.7**	96.9	**73.9**	**91.0**	**92.2**	**92.7**	**90.8**	**88.2**	**91.0**
$\text{Megatron}_{1.3B}$ [49]	1.3B	90.9/91.0	92.6	–	–	–	–	–	–	–	87.1	90.2
$\text{Megatron}_{3.9B}$ [49]	3.9B	91.4/91.4	92.7	–	–	–	–	–	–	–	88.5	91.2

Table 1: Results on GLUE and SQuAD 2.0 development set. All results are single-task, single-model fine-tuning. Results not available in public reports are marked as "–". DeBERTa reported RTE, MRPC and STS-B results by fine-tuning from MNLI checkpoints which are not single-task results. We use Spearman correlation for STS, Matthews correlation for CoLA, and accuracy for the rest on GLUE. AVG is the average of the eight tasks on GLUE. All baseline results unless marked by (Ours) are reported by previous research.

Evaluation Results

Model	Params	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	RTE	MRPC	STS-B	AVG
Base/Base++ Setting: BERT Base Size										
$\text{BERT}_{\text{Base}}$	110M	84.6/83.4	89.2	90.5	93.5	52.1	66.4	84.8	85.8	80.8
$\text{ELECTRA}_{\text{Base++}}$	110M	88.5/88.0	89.5	93.1	**96.0**	64.6	75.2	88.1	90.2	85.6
$\text{COCO-LM}_{\text{Base++}}$	134M	**89.8/89.3**	**89.8**	**94.2**	95.6	**68.6**	**82.3**	**88.5**	**90.3**	**87.4**
Large/Large++ Setting: BERT Large Size										
$\text{BERT}_{\text{Large}}$	335M	86.7/85.9	89.3	92.7	94.9	60.5	70.1	85.4	86.5	83.2
$\text{ELECTRA}_{\text{Large++}}$	335M	90.7/90.2	90.4	95.5	**96.7**	68.1	86.1	**89.2**	91.7	88.5
$\text{COCO-LM}_{\text{Large++}}$	367M	**91.6/91.1**	**90.5**	**95.8**	**96.7**	**70.5**	**89.2**	88.4	**91.8**	**89.3**

Table 2: GLUE test set results obtained from the GLUE leaderboard. We perform hyperparameter search for each task with ten random seeds and use the best development set model for test predictions. All results are from vanilla single-task fine-tuning (no ensemble, task-specific tricks, etc.).

Efficiency

- COCO-LM is more efficient in GPU hours. It outperforms RoBERTa & ELECTRA by 1+ points

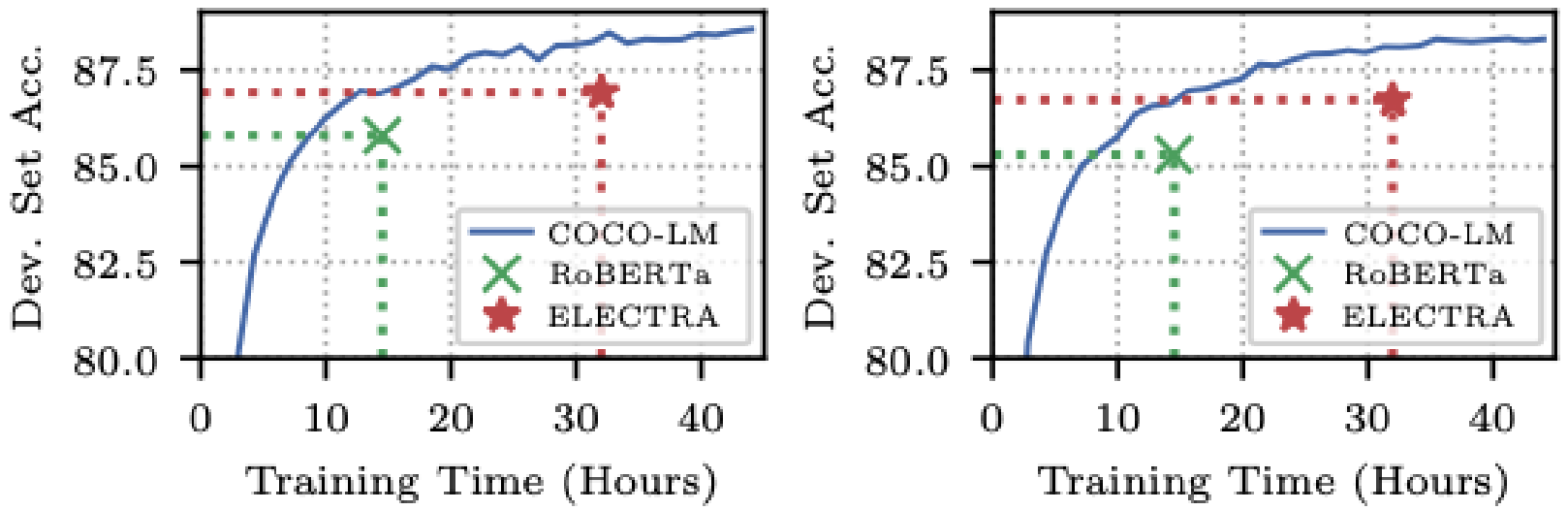


Figure 3: COCO-LM$_{\text{Base}}$ on MNLI Dev. (y-axes) at different pretraining hours on four DGX-2 nodes (64 V100 GPUs). The final training hours and accuracy of RoBERTa (Ours) and ELECTRA (Ours) measured in the same settings are marked.

Ablation Studies

- base setting on GLUE DEV
- CoLA task는 왜 이렇게 점수가 낮게 나왔을까
- SCL의 효과는 0.3 AVG 정도..?! (COCO-LM vs CLM Only -> 근데 막상 STS-B는 CLM Only와 비슷, MRPC는 COCO-LM이 더 높고)

Group	Method	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	RTE	MRPC	STS-B	AVG
	COCO-LM$_{\text{Base}}$	88.5/88.3	92.0	93.1	93.2	63.9	84.8	91.4	90.3	87.2
Pretraining Task	RTD Only	88.4/88.2	92.1	93.5	92.7	67.3	80.5	89.0	90.9	86.8
	CLM Only	88.6/88.4	92.0	93.2	93.7	67.4	80.1	90.0	90.4	86.9
	SCL + RTD	88.6/88.2	92.1	93.5	93.8	64.3	82.7	90.2	90.6	86.9
Network Setting	w/o. Rel-Pos	88.2/87.7	92.2	93.4	93.7	68.8	82.7	91.2	90.6	87.6
	w. ELECTRA's Auxiliary	88.0/87.7	91.9	92.7	93.5	64.3	81.2	89.5	89.7	86.3
Training Signal	w. Random Replacements	84.9/84.7	91.4	91.1	91.4	41.6	70.0	87.3	87.1	80.6
	w. Converged Auxiliary	88.3/88.1	92.0	92.8	94.3	64.2	78.3	90.4	90.2	86.3
CLM Setup	All-Token LM Only	87.2/87.0	91.8	92.6	93.7	60.6	74.0	88.5	89.7	84.7
	CLM w/o. Copy	88.0/87.9	91.8	93.1	94.4	66.6	76.9	89.5	90.1	86.3
	CLM w/o. Stop-grad	88.5/88.2	92.0	92.9	94.3	66.5	80.9	90.0	90.6	86.9

Table 3: Ablations on GLUE Dev. that eliminate (w/o.), keep (Only) or switch (w.) one component.

Architecture.

- Removing relative position encoding (`Rel-Pos`) leads to better numbers on some tasks *but significantly hurts MNLI*.

Pretraining Signal Construction.

- Using `randomly replaced tokens to corrupt text sequence hurts significantly`. Using a converged auxiliary network to pretrain the main model also hurts. It is better to pretrain the two Transformers together

CLM Setup.

- `Disabling the multi-task learning` and using All-Token MLM [7] reduces model accuracy.
- The copy mechanism is effective. The benefits of the stop gradient operation are more on stability (preventing training divergence).

Analyses of Contrastive Learning with SCL

Ablation on Data Augmentation

- original sequence는 성능이 낮아지는데, less informative하기 때문이라고 해석함
    - 문장레벨 변형에 대해 robust하게 배울수가 없어서

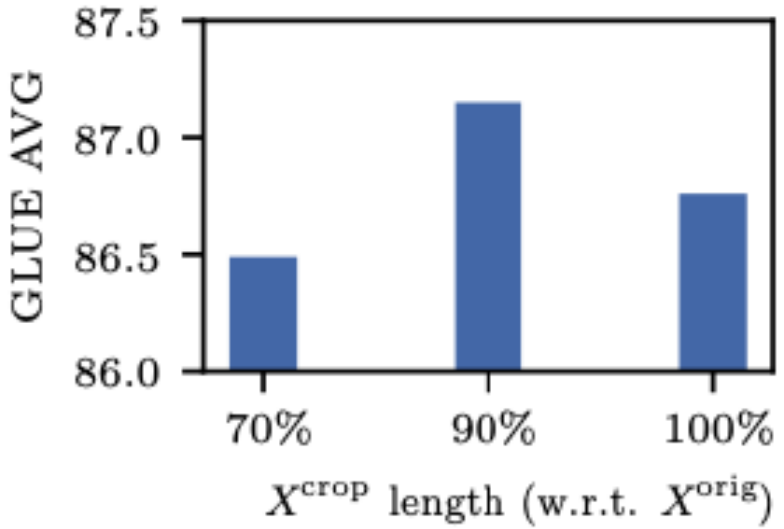
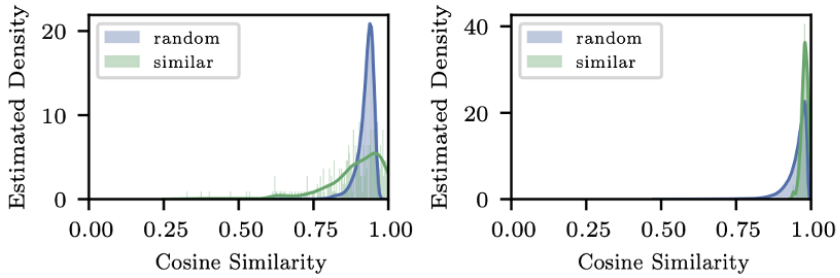


Figure 4: The performance of COCO-LM$_{Base}$ when pretrained with different crop fractions. The x-axis is the fraction of X^{orig} being kept (no cropping is 100%).

Analyses of Contrastive Learning with SCL

Alignment and Uniformity

- The representation space from COCO-LM is drastically different from those in Figure 1
- With COCO-LM, similar pairs are more aligned and random pairs are distributed more uniformly
  - Many `similar pairs have near 1 cosine similarity` and are clearly separated from `random pairs which center around 0`.



(a) RoBERTa. (b) ELECTRA.

Figure 1: Cosine similarity distributions of random/similar sequence pairs using `[CLS]` embeddings from pretrained models. Histograms/curves are distribution bins/kernel density estimates.

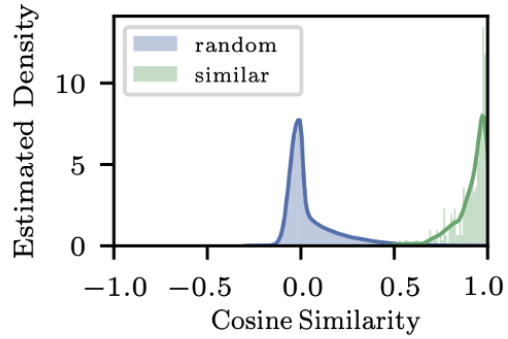


Figure 5: Cosine similarity of sequence pairs randomly sampled from pretraining corpus and most similar pairs from STS-B using `[CLS]` from COCO-LM$_{\text{Base}}$.

Alignment and Uniformity

- t-SNE에서 similiar sentence pairs 의 average cosine similarity is 0.925 when pretrained with SCL, while is 0.863 without SCL. (`Figure 6`)

Regularizing the Representation Learning for Better Few-Shot Ability.

- SCL 적용하면 cosine sim 잘나오고, repsentation이 잘되니 MNLI 기준 성능도 좋아지더라
- SCL is necessary to `regularize the representation space` and to reduce the risk of degeneration

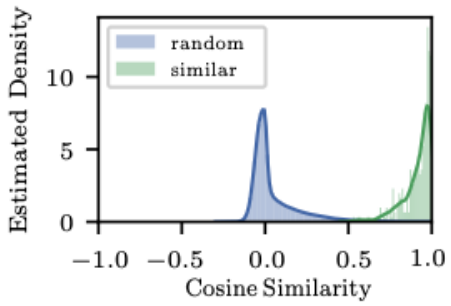
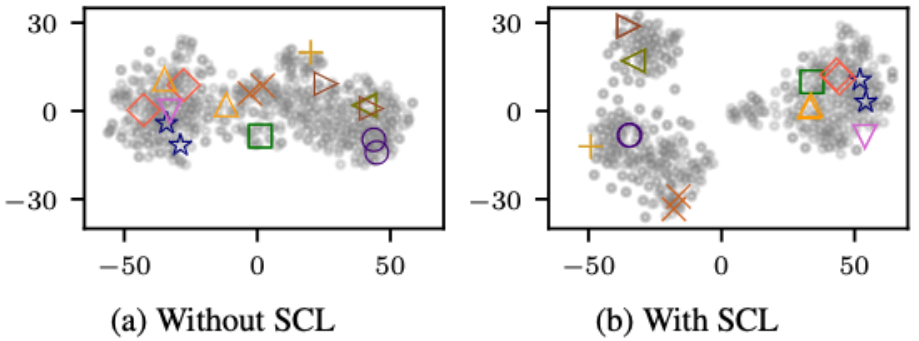
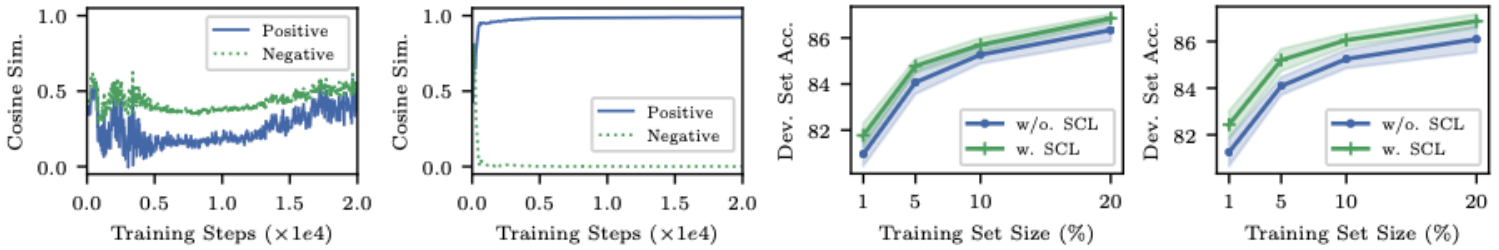


Figure 5: Cosine similarity of sequence pairs randomly sampled from pretraining corpus and most similar pairs from STS-B using `[CLS]` from COCO-LM$_{\text{Base}}$.



(a) Without SCL (b) With SCL

Figure 6: The t-SNE of sequence representations learned with or without SCL. The points are sampled from the most semantically similar sentences pairs from STS-B (with 5-score labels). The `[CLS]` embeddings are not fine-tuned. Some randomly selected similar pairs are marked by same shapes.

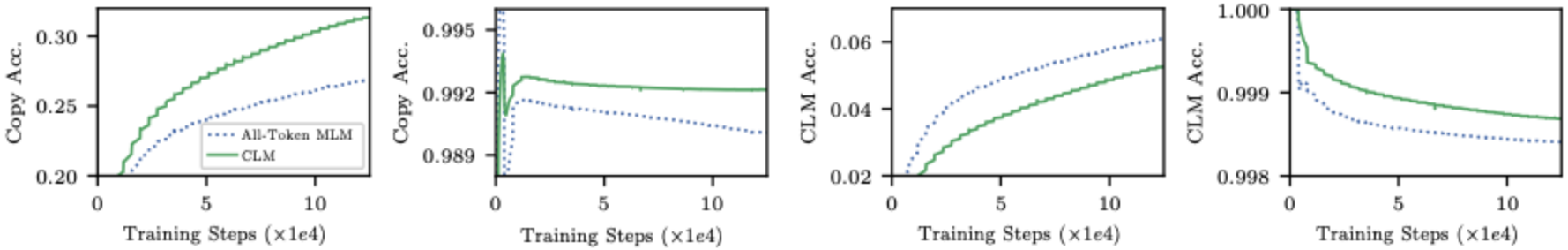


(a) Without SCL (b) With SCL (c) MNLI-m (d) MNLI-mm

Figure 7: Analyses of SCL. Figs. (a) and (b) show the average cosine similarity between the `[CLS]` embeddings of positive and negative contrastive pairs during pretraining. Figs. (c) and (d) show the few-shot accuracy on MNLI with different fractions of MNLI training set used (x-axes). The error bars mark the max/min and the solid lines are the average of five fine-tuning runs.

Analyses of Language Modeling with CLM

- CLM과 All-Token MLM 비교
- It is quite an unbalanced task
  - 대부분의 토큰들(Original)은 단순히 그 토큰을 input으로 copy하는 task를 수행
    - For `the majority of the tokens (Original) the task is simply to copy its input` at the same position.
  - 약 7~8% 정도가 교체된 토큰(Replaced)이고 aux model에서 온 토큰을 orginal로 복구시키는 task를 수행
    - For the replaced tokens (7 – 8% total), however, the model needs to detect the abnormality brought by the auxiliary model and recover the original token
  - Implicitly training the copy mechanism as part of the hard LM task is not effective: `The copy accuracy of All-Token MLM is much lower, and thus the LM head may confuse original tokens with replaced ones`
    - ELECTRA, pretraining with All-Token MLM performs worse than using `the RTD task` (Table 3), though the latter is `equivalent` to only training the `copy mechanism`
    - The `multi-task learning of CLM is necessary` for the main Transformer to stably learn the language modeling task upon the corrupted text sequence.
- (c) 그래프는 잘 이해가 안감



(a) Copy Acc. (Replaced) (b) Copy Acc. (Original) (c) CLM Acc. (Replaced) (d) CLM Acc. (Original)

Figure 8: The copying accuracy and the language modeling accuracy (y-axes) of CLM and All-Token MLM at different pretraining steps (x-axes, in 10K scale). The accuracy is averaged on tokens that are replaced by the auxiliary Transformer (Replaced) or those from the original input text (Original).

Prompt-Based Fine-Tuning with CLM

- RoBERTa 대비 결과가 괜찮은편
- `[MASK]` 토큰 본적이 없는데, 그래도 prompt-base learning을 잘하더라
  - the prompt-based fine-tuning experiments on MNLI for RoBERTa and COCO-LM under base++ and large++ sizes
  - COCO-LM's main Transformer does not even see any [MASK] tokens during pretraining but still performs well on predicting masked tokens for prompt-based learning.
  - Note that ELECTRA and COCO-LM variants without the CLM task are not appli- cable: Their main Transformers are not pretrained by language modeling tasks (thus no language modeling capability is learned to generate prompt label words).

Model	MNLI-m	MNLI-mm
$\text{RoBERTa}_{\text{Base++}}$	60.1 (1.5)	61.8 (1.2)
$\text{COCO-LM}_{\text{Base++}}$	66.5 (2.1)	68.0 (2.3)
$\text{RoBERTa}_{\text{Large++}}$	70.7 (1.3)	72.0 (1.2)
$\text{COCO-LM}_{\text{Large++}}$	72.0 (1.5)	73.3 (1.1)

Table 4: Few-shot prompt-based fine-tuning using RoBERTa and COCO-LM trained on 16 samples per class. Mean (and standard deviation) accuracy results over 5 different splits on MNLI-m/mm are shown.

Conclusion and Future Work

- we present `COCO-LM`, which pretrains language models using `Corrective Language Modeling and Sequence Contrastive Learning upon corrupted text sequences`
- With standard pre-training data and Transformer architectures, COCO-LM `improves the accuracy` on the GLUE and SQuAD benchmarks, while also being `more efficient` in utilizing pretraining computing resources and network parameters
- **One limitation of this work is that the contrastive pairs are constructed by simple cropping and MLM replacements**
- To better understand and tailor the training of the auxiliary model to the main model is another important future research direction

코드구현

- loss 관련 코드 스니펫:
  - https://github.com/microsoft/COCO-LM/issues/2#issuecomment-1003639940
- scl쪽 (span으로 한번 임베딩뽑고, src로도 한번 뽑고)
- 위 코드 위치 (공식 레포에서는 pretraining 관련 코드가 없음, 내부 GPU 셋팅 코드 이슈 때문이라고함):
  - https://github.com/microsoft/COCO-LM/blob/6bb6e5f62d65349657dd51f2f535454a1c50c2e9/fairseq/fairseq/models/cocolm/model.py#L190
- unofficial implementation:
  - https://github.com/lucidrains/coco-lm-pytorch/blob/main/coco_lm_pytorch/coco_lm_pytorch.py

```
def forward(self, src_tokens, span_tokens=None, features_only=False,
            return_all_hiddens=False, classification_head_name=None,
            masked_tokens=None, targets=None, **kwargs):
            # 생략
        if seq_contrast and self.args.span > 0:
            assert span_tokens is not None
            span_padding_mask = get_padding_mask(span_tokens)
            _, extra = self.encoder(
                span_tokens,
                features_only=True,
                return_all_hiddens=return_all_hiddens,
                padding_mask=span_padding_mask,
                seq_contrast=seq_contrast,
                **kwargs
            )
            span_seq_emb = extra["seq_emb"]

    gen_x, extra = self.encoder(
        src_tokens,
        features_only=features_only,
        return_all_hiddens=return_all_hiddens,
        padding_mask=padding_mask,
        masked_tokens=masked_tokens,
        seq_contrast=seq_contrast,
        **kwargs
    )

    if span_seq_emb is not None:
        extra["span_seq_emb"] = span_seq_emb
```