PEER: Pre-training ELECTRA Extended by Ranking

Ru He, Wei Wang, Songfang Huang, Fei Huang
Alibaba Group
{ru.he, hebian.ww, songfang.hsf, f.huang}@alibaba-inc.com

Abstract

The BERT model and its variants have made great achievements in many downstream natural language processing tasks. The achievements of these models, however, demand highly expensive pre-training computation cost. To address this pre-training efficiency issue, the ELECTRA model is proposed to use a discriminator to perform replaced token detection (RTD) task, that is, to classify whether each input token is original or replaced by a generator. The RTD task performed by the ELECTRA accelerates pre-training so substantially, such that it is very challenging to further improve the pre-training efficiency established by the ELECTRA by using or adding other pre-training tasks, as the recent comprehensive study of Bajaj et al. (2022) summarizes. To further advance this pre-training efficiency frontier, in this paper we propose to extend the RTD task into a task of ranking input tokens according to K different quality levels. Essentially, we generalize the binary classifier in the ELECTRA into a K-level ranker to undertake a more precise task with negligible additional computation cost. Our extensive experiments show that our proposed method is able to outperform the state-of-the-art pre-training efficient models including ELECTRA in downstream GLUE tasks given the same computation cost.

1 Introduction

Language model pre-training has made great achievements in many natural language processing (NLP) downstream tasks, by designing and using effective pre-training tasks. A milestone is the BERT model, which conducts a masked language model (MLM) task by randomly masking a proportion (typically 15%) of tokens in the input sentence and then recover the original sentence. Since the success of the BERT, many variant models (Liu et al., 2019; Joshi et al., 2019; Wang et al., 2019; Yang et al., 2019; Dong et al., 2019; Wu et al., 2020) have been proposed to further improve the performance of the BERT by refining or adding pre-training tasks.

One common issue of these MLM-based models, however, is the pre-training efficiency, because they all need highly expensive pre-training computation cost to achieve good performance. To address this issue, the ELECTRA model is proposed by Clark et al. (2020b). The ELECTRA uses an auxiliary generator network to provide plausible tokens to replace a proportion (typically 15%) of the original tokens according to the input context, and then utilizes a main discriminator network to perform replaced token detection (RTD) task, that is, to classify whether each token is original or replaced by the generator. In order to prevent the generator from producing replaced tokens overchallenging for the training of discriminator, the ELECTRA make the generator relative weaker than the discriminator by decreasing the hidden size of the generator. After its pre-training, the generator is discarded and the discriminator is further fine-tuned for downstream NLP tasks. The ELECTRA has shown impressive advantages over MLM-based models in various downstream tasks under similar computation cost, especially when a model size is small.

After the success of the ELECTRA, researchers have proposed quite a few models each of which has an auxiliary generator network. Because the RTD task performed by the ELECTRA has accelerated pre-training so substantially, however, it is very challenging to advance the efficiency frontier established by the ELECTRA, by using or adding other pre-training tasks, as the recent comprehensive study of Bajaj et al. (2022) summarizes. Thus, to further improve the pre-training efficiency, we propose to extend the RTD task in the ELECTRA into a token quality ranking (TQR) task, a task of ranking input tokens according to K different quality levels. Besides determining whether each input token is replaced by a generator or not, the

Findings of the Association for Computational Linguistics: ACL 2023, pages 6475–6491

TQR task also needs to distinguish replaced tokens by ranking them according to their replacement quality. We call our method PEER, Pre-training ELECTRA Extended by Ranking. Please refer to Figure 1 for demonstration. Our proposal is based on the key observation that the quality of replaced tokens are not even. While some replaced tokens fit the context nearly as well as the corresponding original tokens, others do not. Thus, our PEER generalizes the binary classifier in the ELECTRA into a K-level ranker to perform a more precise task. We design a scheme capable of retrieving rank labels for a majority of replaced tokens from the relative weak generator, which serves as the basis for the TQR task. The extension from the ELECTRA to the PEER also adds negligible computation cost, because the TQR task largely re-uses the computation already performed by the original ELECTRA. Additionally, our PEER adopts partial transformer-layer sharing technique between generator and ranker to further reduce computation cost in our method, as its advantage has been demonstrated in the TEAMS model (Shen et al., 2021), a recent model proposed to improve the ELECTRA. Our extensive experiments in small and base scale models show that the PEER is able to outperform both the ELECTRA and the TEAMS in downstream GLUE tasks using the same or less computation cost.

2 Related Work

As introduced in Section 1, since ELECTRA (Clark et al., 2020b) greatly boosts the pre-training efficiency, a few models have been proposed in order to further advance this pre-training efficiency frontier.

The Electric model is proposed by Clark et al. (2020a) as an energy-based model to perform the cloze task (Taylor, 1953) using noise-contrastive estimation (Gutmann and Hyvärinen, 2012). It is particularly effective at producing likelihood scores for text but slightly under-performs ELECTRA on the GLUE tasks.

The MC-BERT model is proposed by Xu et al. (2020) to replace the RTD binary classification task in ELECTRA with a multi-choice cloze test with a reject option (which is essentially a multi-class classification task). The MC-BERT consists of a meta controller network and a generator network. The meta controller corrupts the original input sentence by replacing a proportion of tokens with sampled tokens, just as ELECTRA's generator does. Meanwhile, the meta controller also generates a set of k candidate tokens for each token in the input sentence. The generator uses the corrupted sentence as the input and learns to correct each token by choosing the correct answer among its k candidates. Xu et al. (2020) empirically show that the overall performance of the MC-BERT is similar to that of the ELECTRA in GLUE tasks, since the MC-BERT outperforms the ELECTRA in GLUE semantic tasks but is worse than the ELECTRA in the GLUE syntactic task CoLA.

COCO-LM (Meng et al., 2021) is proposed to improve ELECTRA by using two new pre-training tasks called corrective language modeling (CLM) task and sequence contrastive learning (SCL) task. While ELECTRA's main network (discriminator) conducts only RTD task for each token position, COCO-LM's main network undertakes the CLM task by jointly performing both RTD task and MLM task for each token position in the corrupted input. Additionally, COCO-LM's main network also performs the SCL task to find a pair of the MLM replaced sentence and the cropped sentence originated from the same source sentence among all other sentences in the same training batch.

The DeBERTaV3 (He et al., 2021) is proposed to combine both the advantages of the DeBERTa model (He et al., 2020) and those of the ELECTRA. The DeBERTa (He et al., 2020) introduces two novel mechanisms to improve the effectiveness of the MLM task: disentangled attention and an enhanced mask decoder. The disentangled attention computes the attention weights among tokens using disentangled matrices on two separate vectors (content vector and relative position vector) of each token, while an enhanced mask decoder includes absolute positions in the decoding layer to predict the masked tokens. The DeBERTaV3 keeps these mechanisms but replaces the MLM task (used in the DeBERTa) with ELECTRA's RTD task, and shows that the new combination outperforms both the original DeBERTa and the ELECTRA. Additionally, the DeBERTaV3 introduces gradient-disentangled embedding sharing method as a better alternative to the vanilla token embedding sharing used in the ELECTRA.

The SAS (self-augmentation strategy) is proposed by Xu et al. (2021) in order to improve ELECTRA's pre-training efficiency from the perspective of data augmentation. The SAS uses a sin-

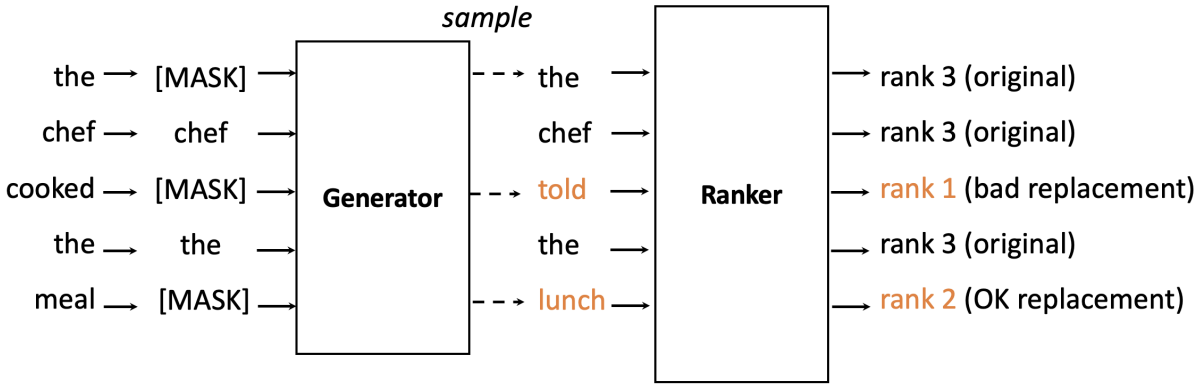

Figure 1: Demonstration of PEER with a 3-level ranker.

gle network to jointly conduct MLM and RTD tasks in order to reduce computation cost and regularize the model parameters for training balance. Essentially, the generator and the discriminator share all their transformer layers in the SAS, and only two separate light-weight heads (MLM and RTD heads) are built on top of the common heavy-weight transformer layers. The MLM head also samples one token in each selected position in order to generate the corrupted input used for the next epoch of the pre-training. The SAS is empirically shown by Xu et al. (2021) to outperform the ELECTRA in small models in GLUE tasks given the same computation cost, but such an advantage vanishes in larger models.

The TEAMS is proposed by Shen et al. (2021) to improve the ELECTRA by adding a multi-word selection (MWS) task along with the original RTD task. Similar to the MC-BERT, the MWS task, which is a multi-choice cloze test, is conducted to choose one correct answer from a candidate set of tokens provided from the generator. Different from the MC-BERT, however, the candidate set in the TEAMS does not contain a reject option, since the MWS task is only performed at the masked positions (instead of all positions). Besides adding the MWS task, the TEAMS introduces two refinements to model structure. One is to share bottom transformer layers of the generator and the discriminator, the other is to use separate top transformer layers for RTD head and MWS head. Both refinements have been empirically shown to be able to further improve the performance of the TEAMS.

Recently, Bajaj et al. (2022) conduct a comprehensive empirical study of ELECTRA-style pre-training techniques, and propose a corresponding pre-training recipe for Model-generated dEnoising TRaining Objective (METRO). Their pre-training recipe incorporates a set of techniques to improve the efficiency and stability of large scale model pre-training, such as the ZeRO optimizer (Rajbhandari et al., 2020), scaled initialization techniques, customized Fused Operations in mix-precision training. In terms of pre-training tasks, however, the empirical study by Bajaj et al. (2022) shows that many previously proposed tasks, such as multi-choice cloze test (Xu et al., 2020), CLM and SCL (Meng et al., 2021), do not provide much improvement for the RTD task in GLUE and SQuAD tasks.

3 Method

In this section, we describe our PEER method, which extends the binary discriminator of the ELECTRA into a ranker. Our PEER method jointly trains two neutral networks, an auxiliary generator network G and a main ranker network R. Each network is mainly a Transformer encoder (Vaswani et al., 2017), which transforms an input token sequence $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ into a sequence of contextualized representation vectors $h(\boldsymbol{x}) = (h(\boldsymbol{x})_1, h(\boldsymbol{x})_2, \cdots, h(\boldsymbol{x})_n)$.

3.1 Generator in PEER

The generator G in the PEER works exactly the same as the generator in the ELECTRA. It first randomly selects a proportion (typically 15%) of position indexes $\{1, \cdots, n\}$ to produce a masked position set $\mathcal{M}$. It then generates a masked token sequence $\boldsymbol{x}^M$ by replacing x_i in $\boldsymbol{x}$ with a special mask token [MASK] for each $i \in \mathcal{M}$. Afterwards, the generator G transforms the input $\boldsymbol{x}^M$ into $h_G(\boldsymbol{x}^M)$ through transformer layers. For position i, the token generating probability of any token x_v given the context $\boldsymbol{x}^M$ is produced from a

softmax function as follows:

$$p_G^{(i)}(x_v|\boldsymbol{x}^M) = \frac{\exp\{e(x_v)^T h_G(\boldsymbol{x}^M)_i\}}{\sum_{x' \in V} \exp\{e(x')^T h_G(\boldsymbol{x}^M)_i\}}, \quad (1)$$

where $e(x_v)$ is the embedding of token x_v, and V is the vocabulary. The inner product $e(x_v)^T h_G(\boldsymbol{x}^M)_i$ in E.q. (1) is essentially a logit of token x_v at position i given the context $\boldsymbol{x}^M$, denoted as

$$\text{logit}^{(i)}(x_v|\boldsymbol{x}^M) := e(x_v)^T h_G(\boldsymbol{x}^M)_i, \quad (2)$$

which will be also used in our ranker.

The loss of the MLM task $\mathcal{L}_{MLM}(\boldsymbol{x}; \theta_G)$ is a cross entroy loss (i.e., negative log likelihood):

$$\mathcal{L}_{MLM}(\boldsymbol{x}; \theta_G) = -\sum_{i \in \mathcal{M}} \log p_G^{(i)}(x_i|\boldsymbol{x}^M).$$

For each position i in $\mathcal{M}$, the generator also sample one token $\hat{x}_i$ from the token generating probability $p_G^{(i)}(\cdot|\boldsymbol{x}^M)$, and then replace the original token x_i with the sampled $\hat{x}_i$ to produce a corrupted token sequence $\boldsymbol{x}^C$.

3.2 Ranker in PEER

Given the corrupted token sequence $\boldsymbol{x}^C$, the K-level ranker performs token quality ranking (TQR) task, that is, assigns each token in $\boldsymbol{x}^C$ into a rank value $r \in \{1, 2, \ldots, K\}$.

Assuming that rank label R_i at position i of the corrupted token sequence $\boldsymbol{x}^C$ is given, for rank value $r \in \{1, 2, \ldots, K-1\}$, the probability that $R_i \leq r$ is given:

$$P(R_i \leq r|\boldsymbol{x}^C) = \sigma(-w^T h_R(\boldsymbol{x}^C)_i + \xi_r), \quad (3)$$

where σ is a sigmoid function, $h_R(\boldsymbol{x}^C)_i$ is the contextualized representation vector at position i out of the ranker transformer, w is the to-be-learned weight vector, $\{\xi_1, \xi_2, \ldots, \xi_{K-1}\}$ is a set of to-be-learned threshold parameters with the property $\xi_1 < \xi_2 < \cdots < \xi_{K-1}$.

The binary discriminator in the ELECTRA can be viewed as a ranker with $K = 2$, where rank label R_i is naturally given by

$$\begin{cases} R_i = 1 & \text{if } x_i^c \neq x_i \\ R_i = 2 & \text{if } x_i^c = x_i \end{cases}$$

where x_i^c is the token at position i in $\boldsymbol{x}^C$.

Accordingly, the loss of the TQR task with $K = 2$ levels is a binary cross entropy loss:

$$\begin{aligned} &\mathcal{L}_{TQR}(\boldsymbol{x}; \theta_R) \\ &= -\Bigg[\sum_{i=1}^{n} \Big(I[R_i \leq K-1] \cdot \\ &\quad \log P(R_i \leq K-1|\boldsymbol{x}^C) \\ &\quad + I[R_i > K-1] \log P(R_i > K-1|\boldsymbol{x}^C)\Big)\Bigg], \end{aligned} \quad (4)$$

where $I[]$ is an indicator function.

3.2.1 Rank Label Retrieving Scheme

In order to use a ranker with $K > 2$, we need to assign a rank label R_i to each x_i^c, the token at position i in $\boldsymbol{x}^C$. Thus, we design a label retrieving scheme to obtain rank labels from the generator. For notational convenience, we use $p_o^{(i)}$ to denote $p_G^{(i)}(x_i|\boldsymbol{x}^M)$, the generating probability of the original token x_i at position i in the context; and use $p_c^{(i)}$ denote $p_G^{(i)}(x_i^c|\boldsymbol{x}^M)$, the generating probability of any token x_i^c at position i in the context. We use $\text{rank}(p_o^{(i)}) \leq T$ to represent that $p_o^{(i)}$ is within top T out of all $|V|$ probabilities from $p_G^{(i)}(\cdot|\boldsymbol{x}^M)$, where T is a hyperparameter with a small value[1].

Our rank label retrieving scheme is shown in Table 1, where $\{\tau_1, \tau_2, \cdots, \tau_{K-2}\}$ are a set of probability partitioning hyperparameters with property $0 < \tau_1 < \tau_2 < \cdots < \tau_{K-2}$.[2] We set the rank label of the original token to the highest value K. For each replaced token x_i^c (which differs from x_i), we set up the levels (buckets) based on $p_o^{(i)}$ and $\{\tau_1, \tau_2, \cdots, \tau_{K-2}\}$, so that the rank label of x_i^c is set according to the bucket which $p_c^{(i)}$ will fall into.

Note, however, just as the ELECTRA, the generator in our PEER is set to be weak (small) relative to the ranker in order to prevent generating too-challenging replaced tokens. Therefore we use the condition $\text{rank}(p_o^{(i)}) \leq T$ to identify every position i where the generator can provide well-estimated probability $p_G^{(i)}(\cdot|\boldsymbol{x}^M)$ for tokens x_i and x_i^c. For every replaced token x_i^c at position i where $\text{rank}(p_o^{(i)}) > T$, we just set its rank label to a special value -1 to indicate that its rank is less than K but the exact rank value is unknown.[3]

[1] We set T to 3 in our experiments.

[2] We always set τ_1 to 1 for a ranker with $K > 2$ in our experiments.

[3] Appendix A.3 will show that a majority of tokens in the masked positions have their rank labels other than -1 when T is set to 3 in both small and base models.

R_i	Condition
K	$x_i^c = x_i$
$K-1$	$p_c^{(i)} \in [p_o^{(i)}/\tau_1, 1] \wedge x_i^c \neq x_i \wedge \text{rank}(p_o^{(i)}) \leq T$
$K-2$	$p_c^{(i)} \in [p_o^{(i)}/\tau_2, p_o^{(i)}/\tau_1) \wedge x_i^c \neq x_i \wedge \text{rank}(p_o^{(i)}) \leq T$
$\cdots$	$\cdots$
2	$p_c^{(i)} \in [p_o^{(i)}/\tau_{K-2}, p_o^{(i)}/\tau_{K-3}) \wedge x_i^c \neq x_i \wedge \text{rank}(p_o^{(i)}) \leq T$
1	$p_c^{(i)} \in [0, p_o^{(i)}/\tau_{K-2}) \wedge x_i^c \neq x_i \wedge \text{rank}(p_o^{(i)}) \leq T$
-1	$x_i^c \neq x_i \wedge \text{rank}(p_o^{(i)}) > T$

Table 1: Rank label retrieving scheme for a K-level ranker in PEER.

R_i	Condition
K	$x_i^c = x_i$
$K-1$	$\text{logit}^{(i)}(x_i^c\|\boldsymbol{x}^M) - \text{logit}^{(i)}(x_i\|\boldsymbol{x}^M) \in [-\log\tau_1, \infty) \wedge x_i^c \neq x_i \wedge \text{rank}(p_o^{(i)}) \leq T$
$K-2+\Delta$	$\text{logit}^{(i)}(x_i^c\|\boldsymbol{x}^M) - \text{logit}^{(i)}(x_i\|\boldsymbol{x}^M) \in [-\log(\tau_1(1+\delta)), -\log\tau_1) \wedge x_i^c \neq x_i \wedge \text{rank}(p_o^{(i)}) \leq T$
$K-2$	$\text{logit}^{(i)}(x_i^c\|\boldsymbol{x}^M) - \text{logit}^{(i)}(x_i\|\boldsymbol{x}^M) \in [-\log\tau_2, -\log(\tau_1(1+\delta)) \wedge x_i^c \neq x_i \wedge \text{rank}(p_o^{(i)}) \leq T$
$\cdots$	$\cdots$
2	$\text{logit}^{(i)}(x_i^c\|\boldsymbol{x}^M) - \text{logit}^{(i)}(x_i\|\boldsymbol{x}^M) \in [-\log\tau_{K-2}, -\log(\tau_{K-3}(1+\delta))) \wedge x_i^c \neq x_i \wedge \text{rank}(p_o^{(i)}) \leq T$
$1+\Delta$	$\text{logit}^{(i)}(x_i^c\|\boldsymbol{x}^M) - \text{logit}^{(i)}(x_i\|\boldsymbol{x}^M) \in [-\log(\tau_{K-2}(1+\delta)), -\log\tau_{K-2}) \wedge x_i^c \neq x_i \wedge \text{rank}(p_o^{(i)}) \leq T$
1	$\text{logit}^{(i)}(x_i^c\|\boldsymbol{x}^M) - \text{logit}^{(i)}(x_i\|\boldsymbol{x}^M) \in (-\infty, -\log(\tau_{K-2}(1+\delta)))) \wedge x_i^c \neq x_i \wedge \text{rank}(p_o^{(i)}) \leq T$
-1	$x_i^c \neq x_i \wedge \text{rank}(p_o^{(i)}) > T$

Table 2: Rank label retrieving scheme internally implemented for a K-level ranker in PEER, along with an additional buffer option associated with hyperparameter δ.

For the purpose of numerical stability, determining which bucket $p_c^{(i)}$ falls into (in Table 1) is actually implemented using its equivalent form on the basis of the logit difference: $\text{logit}^{(i)}(x_i^c|\boldsymbol{x}^M) - \text{logit}^{(i)}(x_i|\boldsymbol{x}^M)$, where both logit terms are defined in E.q. (2). Additionally, in Table 2 we also introduce a buffer option between level k and level $k+1$ for each $k \in \{1, \cdots, K-2\}$ to further safe-guard against the relative weakness of the generator. All the data points in the buffer are regarded as being in a grey (potentially noisy) area and are excluded for the binary classification between level k and level $k+1$. If we want to use the buffer option, we add a positive hyperparameter δ inside the relevant buckets[4] to set up the buffers and add a small fixed value $\Delta \in (0, 1)$ to the corresponding rank label[5]. A larger value of δ leads to the smaller number of the training data points, but adds confidence in removing potentially noisy data points. If we do not want to use the buffers, we set hyperparameter δ equal to 0 so that these buffers will disappear.

[4]The hyperparameter δ needs to satisfy the condition that $\tau_k(1+\delta) \leq \tau_{k+1}$ for each $k \in \{1, \cdots, K-3\}$.

[5]We internally set Δ to 0.1 though its exact value does not matter.

3.2.2 Loss of TQR Task

Because some replaced tokens have their exact rank labels unknown (represented by the special value -1), the loss of the TQR task cannot be directly formulated as the loss of standard ordinal regression (McCullagh and Nelder, 1989). To address this challenge, we set the loss of the TQR task with K levels to be the summation of $K-1$ binary cross entropy losses:

$$
\begin{aligned}
&\mathcal{L}_{TQR}(\boldsymbol{x}; \theta_R) \\
&= -\Bigg[\sum_{i=1}^{n}\Big(I[R_i \leq K-1]\cdot \\
&\quad \log P(R_i \leq K-1|\boldsymbol{x}^C) \\
&\quad + I[R_i > K-1]\log P(R_i > K-1|\boldsymbol{x}^C)\Big) + \\
&\quad \sum_{r=1}^{K-2}\gamma_r \sum_{\substack{i\in\{1,\cdots,n\} \\ i:R_i\neq -1}}\Big(I[R_i \leq r]\log P(R_i \leq r|\boldsymbol{x}^C) \\
&\quad + I[R_i > r+\Delta]\log P(R_i > r|\boldsymbol{x}^C)\Big)\Bigg],
\end{aligned}
\tag{5}
$$

where γ_r is a positive relative weight hyperparameter for the binary cross entropy loss at level r [6] Essentially, $\mathcal{L}_{TQR}$ contains both the loss of RTD task stated in E.q. (4) and each binary entropy loss at level $r \in \{1, \cdots, K-2\}$.

We set the loss of the TQR task to the summation of $K-1$ binary cross entropy losses in E.q. (5) in the entire pre-training process except a beginning warming-up phase. In the warming-up phase[7] we still use only one binary cross entropy stated in E.q. (4) as the loss of the TQR task, in order to ensure that the generator gets some basic training so that its token generating probability $p_G^{(i)}(\cdot|\boldsymbol{x}^M)$ is generally reliable for the rank labeling purpose.

Overall, we train the PEER by minimizing a combined loss:

$$\sum_{\boldsymbol{x}\in\mathcal{X}} \mathcal{L}_{MLM}(\boldsymbol{x};\theta_G) + \lambda\mathcal{L}_{TQR}(\boldsymbol{x};\theta_R),$$

where λ is the relative weight for the loss of TQR task[8]. After pre-training, we discard the generator and fine-tune the ranker for downstream NLP tasks.

As an additional note, extending the ELECTRA to the PEER requires negligible increase in computation cost. The only added parameters in our PEER are $K-1$ threshold parameters $\{\xi_1, \xi_2, \ldots, \xi_{K-1}\}$. The same sequence contextualized representation vectors $h_R(\boldsymbol{x}^C)$ out of the ranker transformer is re-used for different levels, along with the shared weight parameter vector w in E.q. (3). The R_i labeling is also based on the logit information in E.q. (2) which has already been computed for $p_G^{(i)}(\cdot|\boldsymbol{x}^M)$ in the generator.

4 Experiments

4.1 Experimental Setup

Pre-training Details: We implement the PEER within Huggingface Transformers framework (Wolf et al., 2020). We include ELECTRA, TEAMS as well as BERT for comparison. Under the current constraints of computation resource, we focus on the small and base models which have been extensively studied and compared by Clark et al. (2020b), and we set architectures and hyperparameters largely aligned with ELECTRA. Please refer to Appendix B for the detailed model architecture and pre-training hyperparameter values. We implement each model by largely re-using the corresponding code from Huggingface (Wolf et al., 2020), if a pre-trained checkpoint has not been publicly released by its authors. We use the same pre-training data as BERT, ELECTRA-Small and ELECTRA-Base, which consists of 3.3 Billion tokens from Wikipedia and BooksCorpus datasets. For fair comparison, we follow Clark et al. (2020b) to use FLOPs (floating point operations) to measure computation usage (since FLOPs is a measure agnostic to the particular hardware and low-level optimizations). We reuse the FLOPs computation code[9] released from Clark et al. (2020b) so that we essentially take the exactly same assumptions made by Clark et al. (2020b). Some details of the experimented models are as follows.

- **ELECTRA**: We pre-train ELECTRA-Small and ELECTRA-Base using the exactly same hyperparameter values as Clark et al. (2020b), except for larger batch size and learning rate for ELECTRA-Small to reduce the pre-training time (which is not reflected in the FLOPs calculation). For ELECTRA-Small model as well as all other small models, we use batch size 512 and 250K pre-training steps, instead of batch size 128 and 1M steps in Clark et al. (2020b). Accordingly, we add 100% increase in learning rate for ELECTRA-Small and BEET-Small, and add 50% increase in learning rate for TEAMS-Small and PEER-Small [10]. We observe that the change in batch size and learning rate is able to significantly reduce the pre-training time without degrading the model performance. As a reference point, we also include ELECTRA-Small++ whose pre-trained model checkpoint is publicly released by Clark et al. (2020b). Note that ELECTRA-Small++ uses 18x training FLOPs compared to ELECTRA-Small, because it is pre-trained much longer with much larger data and its input sequence length is also quadrupled (Clark et al., 2020b).

- **BERT**: For BERT-Base, we use its model checkpoint publicly released by Devlin et al. (2018). We implement our BERT-Small and set its embedding size the same as its hid-

[6] We set every γ_r to 0.5 in our experiments.

[7] We use the first 0.8 epoch as the warming-up phase in our experiments.

[8] We set λ to 50 in our experiments.

[9] See https://github.com/google-research/electra/blob/master/flops_computation.py

[10] We find that 100% increase in learning rate for TEAMS-Small or PEER-Small often leads to some overflow errors during their pre-training.

den size[11], according to the convention of the BERT models. Please refer to the appendix for the details about the hyperparameters. Our BERT-Small setting makes its FLOPs similar to that of ELECTRA-Small when the training steps are the same, so that fair comparison of their performance can be made directly.

- **TEAMS**: We pre-train TEAMS-Small and TEAMS-Base using the same hyperparameter values described by Shen et al. (2021), except the aforementioned larger batch size and learning rate. The model structures of TEAMS-Small and TEAMS-Base are also the same as the ones used by Shen et al. (2021). Specifically, the discriminator in TEAMS-Small has 12 transformer layers and set its hidden size to 256; and the discriminator in TEAMS-Base has 12 transformer layers and set its hidden size to 768. The generator has 6 transformer layers and set its hidden size same as the corresponding discriminator. The generator and the discriminator share three layers on the bottom, the discriminator also has one additional separate transformer layer on the top for its MWS task.

- **PEER**: We pre-train PEER using the hyperparameter values the same as TEAMS. With respect to hyperparameter δ, we set it to 3 in PEER-Small and set it to 9 in PEER-Base for model comparison, and discuss the effect of δ in Appendix A.2 due to space constraint. We focus on the PEER with 3-level ranker, and discuss the effect of the number of levels in Appendix A.1. The model structures of the PEER are the same as the TEAMS, except that there is no additional transformer layer for MWS task in the PEER. This difference makes FLOPs per training step in the PEER smaller than the ones of the corresponding TEAMS. However, FLOPs per training step in the PEER are still larger than the ones of the corresponding ELECTRA. This is largely because the generator in the ELECTRA decreases its hidden size (instead of its number of transformer layers), which in turn leads to the decrease in the intermediate size in every fully connected feed-forward network (FFN). We will clearly record these training FLOPs in our experimental results and ensure that our PEER uses FLOPs no more than other models during the performance comparison.

[11] Clark et al. (2020b) define a different BERT-Small setting where its embedding size is decreased to half of its hidden size.

Downstream Tasks and Metrics: We evaluate all models on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018). It contains a variety of tasks covering natural language inference tasks MNLI (Williams et al., 2017), QNLI (Rajpurkar et al., 2016) and RTE (Giampiccolo et al., 2007); semantic similarity tasks MRPC (Dolan and Brockett, 2005), QQP (Iyer et al., 2017), and STS-B (Cer et al., 2017); sentiment classification task SST-2 (Socher et al., 2013); and linguistic acceptability classification CoLA (Warstadt et al., 2019). See Appendix C.1 for more details on the GLUE tasks.

The evaluation metrics are the average of MNLI-match accuracy and MNLI-mismatch accuracy for MNLI, the average of Spearman correlation and Pearson correlation for STS-B, Matthews correlation for CoLA, and accuracy for other GLUE tasks. We also take the average of metrics of these eight GLUE tasks, denoted by G-AVG, as the overall performance metric on these tasks. All the evaluation is based on the Dev dataset.

Fine-tuning Procedure: For the fine-tuning of GLUE tasks, we add simple linear classifiers on top of the encoder of a pre-trained model. Because we observe a large performance variance in the GLUE tasks with small data sizes (including CoLA, MRPC, STS-B and RTE), we adopt the following two methods to reduce the variance. First, we follow the strategy proposed in the papers (Mosbach et al., 2020; Zhang et al., 2020; Dodge et al., 2020) to train more epochs with small learning rates for these small tasks. Second, we fine-tune these small tasks by using multiple random seeds and obtain the average score across the seeds. Please refer to Appendix C for the details in fine-tuning hyperparameter settings.

For base models, we pre-train each model once and then use the above fine-tuning strategy to obtain the score of each GLUE task. Since for some small models we still observe non-negligible variance of the resulting scores, we pre-train each small model using five different random seeds. The finally reported score of each task is the average across the five pre-trained model checkpoints.

Model	Train FLOPs	G-AVG Mean±Std	MNLI	CoLA	SST-2	MRPC	STS-B	QQP	QNLI	RTE
BERT-Small	1.27e18	79.11±0.08	79.97	49.53	**90.09**	84.52	86.15	89.57	86.79	66.23
ELECTRA-Small	1.29e18	80.77±0.16	80.22	59.40	89.19	86.48	86.72	89.93	88.27	65.99
*ELECTRA-Small++**	*2.40e19*	*82.05*	*82.52*	*58.37*	*91.40*	*87.01*	*87.95*	*90.54*	*88.93*	*69.68*
TEAMS-Small	1.55e18	80.84±0.23	81.05	55.83	89.81	87.71	87.34	89.72	88.31	**66.94**
PEER-Small (212.5K)	1.27e18	**81.40±0.25**	**81.08**	**59.70**	89.40	**87.99**	**87.70**	**90.12**	**88.45**	66.71
PEER-Small (250K)	1.50e18	81.50±0.30	81.19	59.66	89.29	88.09	87.66	90.15	88.55	67.44

*: ELECTRA-Small++ is the pre-trained model publicly released by Clark et al. (2020b).

Table 3: Comparison of small models on the GLUE dev set.

Model	Train FLOPs	G-AVG	MNLI	CoLA	SST-2	MRPC	STS-B	QQP	QNLI	RTE
*BERT-Base (1M)**	*6.43e19*	*83.51*	*84.51*	*60.07*	*93.00*	*86.03*	*89.51*	*91.27*	*91.51*	*72.20*
ELECTRA-Base (766K)	6.43e19	86.42	85.96	67.05	92.09	91.05	90.47	91.57	92.10	**81.05**
TEAMS-Base (666.9K)	6.66e19	86.11	86.48	66.30	**93.00**	90.44	90.22	91.38	92.36	78.70
PEER-Base (666.9K)	6.39e19	**86.77**	**86.69**	**68.57**	92.66	**91.18**	**90.92**	**91.78**	**92.57**	79.78

*: BERT-Base is the pre-trained model publicly released by Devlin et al. (2018).

Table 4: Comparison of base models on the GLUE dev set.

4.2 Overall Comparison Results

Table 3 shows the performance comparison among the small models. In the table, the second column lists the training FLOPs of each model, and the third column shows the mean and the standard deviation of the G-AVG for each model across five independently pre-trained checkpoints. We report the performance of each small model pre-trained through 250K steps (i.e., 5 epochs). Additionally, we report the performance of PEER-Small pre-trained exactly after 212.5K steps to ensure that its computation cost is no more than that of any other competitor.

Note that the G-AVG of ELECTRA-Small implemented by us is about 98.44% of that of ELECTRA-Small++ released by Clark et al. (2020b) (80.77 vs. 82.05), which is higher than the 97.87% in Table 8 of the original paper (Clark et al., 2020b). This verifies the correctness of our ELECTRA implementation. As for TEAMS-Small, the G-AVG of TEAMS-Small is slightly higher than that of ELECTRA-Small when they go through the same number of pre-training steps, which is consistent with the comparison results shown by Shen et al. (2021). While Shen et al. (2021) do not report the performance of each individual task, our result shows that TEAMS-Small performs much better in MNLI task but much worse in CoLA task when comparing with ELECTRA-Small.

With respect to our PEER, Table 3 clearly demonstrates its advantages over all the other competitors in small models. Using less computation cost, PEER-Small (212.5K) outperforms both ELECTRA-Small and TEAMS-Small in six out of eight GLUE tasks, as SST-2 and RTE tasks are the only two exceptions. The G-AVG of PEER-Small (212.5K) is 0.63 point higher than that of ELECTRA-Small and is 0.56 point higher than that of TEAMS-Small. Because we have independently run the whole (pre-training and fine-tuning) process five times for each small model, by using the two-sample t test with unequal variances, we can conclude with strong evidence (at the significance level 0.005) that the real mean of G-AVG of our PEER-Small (212.5K) is larger than that of ELECTRA-Small. Similarly, based on the two-sample t test with unequal variances, we can conclude with strong evidence (at the significance level 0.005) that the real mean of G-AVG of our PEER-Small (212.5K) is larger than that of TEAMS-Small.

Table 4 shows the comparison results on the base models. In the first column of the table, we show the pre-training steps of each model and have ensured that PEER-Base takes FLOPs no more than other models. Using less computation cost, PEER-Base achieves the best performance among all the investigated models in six out of eight GLUE tasks, while two exceptions are SST-2 and RTE tasks (just as in small models). Overall, PEER-Base has the highest G-AVG, which is 0.35 point higher than that of ELECTRA-Base and is 0.66 point higher than that of TEAMS-Base.

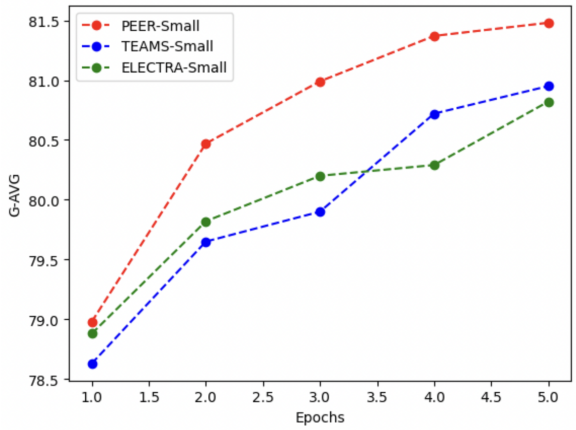

Figure 2: G-AVG for PEER-Small model and its competitors with respect to the number of pre-training epochs on the GLUE dev set.

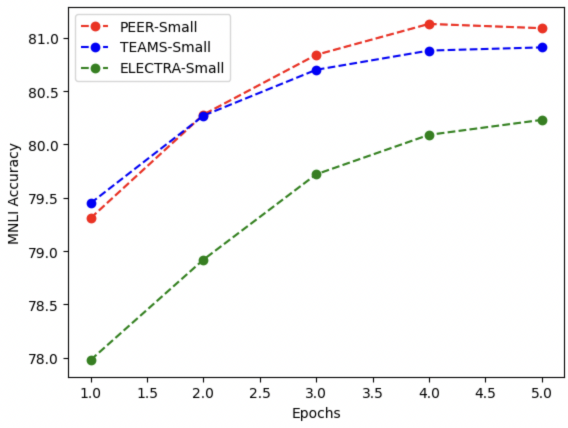

Figure 3: MNLI's average accuracy for PEER-Small model and its competitors with respect to the number of pre-training epochs on the GLUE dev set.

4.3 Pre-Training Efficiency

To further investigate the pre-training efficiency, in Figures 2 and 3, we plot G-AVG and MNLI accuracy score with respect to the number of pre-training epochs for PEER-Small, ELECTRA-Small and TEAMS-Small. For each model, we select the median run whose pre-training random seed achieves the median G-AVG among the five random seeds. Then for the selected median run of each model, we save a checkpoint every epoch (i.e, 50K pre-training steps), and fine-tune it on every GLUE task and finally report the scores across the tasks. Note that the ratio of the training FLOPs per epoch among PEER-Small, ELECTRA-Small and TEAMS-Small is 1.50 : 1.29 : 1.55, which has also been shown in Table 3. Figure 2 shows that PEER-Small starts to significantly outperform its competitors in G-AVG since the second epoch, and its G-AVG at the end of third epoch is already higher than G-AVG of both ELECTRA-Small and TEAMS-Small at the end of the whole pre-training. Figure 3 shows that both PEER-Small and TEAMS-small perform considerably better than ELECTRA-Small in MNLI task, and PEER-Small performs better than TEAMS-small (by using less computation cost) since the third epoch.

5 Conclusion and Future Work

We propose the PEER by extending ELECTRA's RTD task to a token quality ranking (TQR) task in order to further improve the pre-training efficiency. Besides detecting whether every token is replaced or not, the TQR task also needs to rank replaced tokens into different levels according to their quality given the context. We design a scheme to retrieve rank label information from the generator so that the complete TQR task can be performed for a majority of replaced tokens. We empirically show that our proposed PEER outperforms the state-of-the-art pre-training efficient competitors in small and base scale models using the same or less computation cost. In the future, we will validate the advantages of our PEER in larger scale models when sufficient computation resources are available. We also plan to improve our rank label retrieving scheme so that even larger proportion of replaced tokens can be involved in the complete TQR task.

Limitations

There are several limitations in our paper. First, we have not validated the advantages of our proposed PEER in model scales larger than base model, due to the constraint in our computation resource. We plan to experiment the PEER in larger scale models when more computation resource is available. Second, in order to filter out potential noise from the relative weak generator, our current rank label retrieving scheme uses a strict condition $T = 3$, which leads to the fact that a significant proportion of tokens have rank label -1 and essentially are involved only in the original RTD task. Please refer to the details in Appendix A.3. We intend to design some label retrieving scheme which applies a softer criterion so that more tokens can be fully or partially involved in the complete TQR task. Finally, our PEER currently does not have the ability of automatically searching for an optimal value of

hyperparameter δ, which we also plan to design in the future.

References

Payal Bajaj, Chenyan Xiong, Guolin Ke, Xiaodong Liu, Di He, Saurabh Tiwary, Tie-Yan Liu, Paul Bennett, Xia Song, and Jianfeng Gao. 2022. METRO: Efficient denoising pretraining of large scale autoencoding language models with model generated signals. *arXiv preprint arXiv:2204.06644*.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*.

Kevin Clark, Minh-Thang Luong, Quoc Le, and Christopher D. Manning. 2020a. Pre-training transformers as energy-based cloze models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 285–294, Online. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020b. ELECTRA: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*.

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. *CoRR*, abs/1905.03197.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9.

Michael U. Gutmann and Aapo Hyvärinen. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(11):307–361.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTaV3: Improving DeBERTaV using ELECTRA-style pre-training with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. DeBERTa: Decoding-enhanced BERT with disentangled attention. *arXiv preprint arXiv:2006.03654*.

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. 2017. First quora dataset release: Question pairs. *data. quora. com.*

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2019. Spanbert: Improving pre-training by representing and predicting spans. *CoRR*, abs/1907.10529.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

P. McCullagh and J. A. Nelder. 1989. *Generalized Linear Models*. Chapman & Hall / CRC, London.

Yu Meng, Chenyan Xiong, Payal Bajaj, Saurabh Tiwary, Paul Bennett, Jiawei Han, and Xia Song. 2021. COCO-LM: Correcting and contrasting text sequences for language model pretraining. *arXiv preprint arXiv:2102.08473*.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. 2020. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRo: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Jiaming Shen, Jialu Liu, Tianqi Liu, Cong Yu, and Jiawei Han. 2021. Training ELECTRA augmented with multi-word selection. *arXiv preprint arXiv:2106.00139*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Wilson L. Taylor. 1953. "cloze procedure": A new tool for measuring readability. *Journalism Quarterly*, 30(4):415–433.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, page 6000–6010.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461.

Wei Wang, Bin Bi, Ming Yan, Chen Wu, Zuyi Bao, Liwei Peng, and Luo Si. 2019. Structbert: Incorporating language structures into pre-training for deep language understanding. *CoRR*, abs/1908.04577.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Zhuofeng Wu, Sinong Wang, Jiatao Gu, Madian Khabsa, Fei Sun, and Hao Ma. 2020. CLEAR: Contrastive learning for sentence representation. *arXiv preprint arXiv:2012.15466*.

Yifei Xu, Jingqiao Zhang, Ru He, Liangzhu Ge, Chao Yang, Cheng Yang, and Ying Nian Wu. 2021. SAS: Self-augmented strategy for language model pre-training. *arXiv preprint arXiv:2106.07176*.

Zhenhui Xu, Linyuan Gong, Guolin Ke, Di He, Shuxin Zheng, Liwei Wang, Jiang Bian, and Tie-Yan Liu. 2020. MC-BERT: efficient language pre-training via a meta controller. *CoRR*, abs/2006.05744.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. 2020. Revisiting few-sample bert fine-tuning. *arXiv preprint arXiv:2006.05987*.

A Supplementary Experimental Results

A.1 Number of Levels K

We vary K (the number of levels used in the ranker) from 3 to 5 in PEER-Small models to see its impact. Table 5 shows the corresponding results. Each model is pre-trained 212.5K steps and has nearly the same computation cost. The table shows that increasing K from 3 does not lead to further improvement in the performance of GLUE tasks. The G-AVG of the PEER-Small with 4 or 5 levels actually decreases slightly, though it is still larger than that of its competitors shown in Table 3 by using less computation cost. We conjecture that the main reason is that increasing K leads to the smaller number of tokens staying in low levels, which in turn brings difficulty in the learning process. We will further investigate the impact of K in our future work.

A.2 Buffer Hyperparameter δ

We test the impact of buffer hyperparameter δ by using a set of three different values $\{0, 3, 9\}$, where value 0 leads to no buffer and value 9 leads to a large buffer. By its design, a larger buffer leads to the smaller number of the training data points, but adds confidence in removing potentially noisy data points due to the relative weakness of the generator. Tables 6 and 7 show the results in the PEER-Small models and PEER-Base models respectively. Since the value of δ has a negligible effect in the training FLOPs, we do not list the training FLOPs here as they have already been shown in Table 3 and 4. Both tables show that the G-AVG decreases slightly when δ decreases to 0, though it is still no worse than that of any its competing model by using less computation cost. The PEER-Small achieves the highest G-AVG and MNLI scores when δ is set to 3. The PEER-Base achieves the highest G-AVG when δ is set to 9, and achieves the highest MNLI score when δ is set to 3. In the future we will investigate how to let the PEER automatically search for an optimal value of δ during its pre-training to further boost its performance.

A.3 Proportion of Tokens with Rank Label -1

Figures 4 and 5 demonstrate the proportion of tokens with rank label -1 in the masked positions during the pre-training for PEER-Small and PEER-Base. With respect to PEER-Small, the proportion decreases from 44.82% at 40K steps (i.e., the end of the warm-up phase) to 40.42% at the end of the pre-training. With regards to PEER-Base, the proportion decreases from 36.20% at 33344 steps (i.e., the end of the warm-up phase) to 27.33% at the end of the pre-training. Thus, a majority of replaced tokens have their rank labels other than -1 during the pre-training of both PEER-Small and PEER-Base.

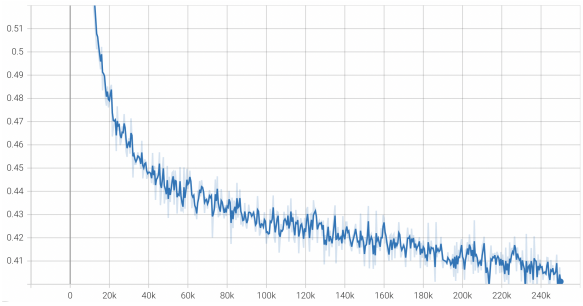

Figure 4: Proportion of tokens with rank label -1 in the masked positions for PEER-Small during the pre-training.

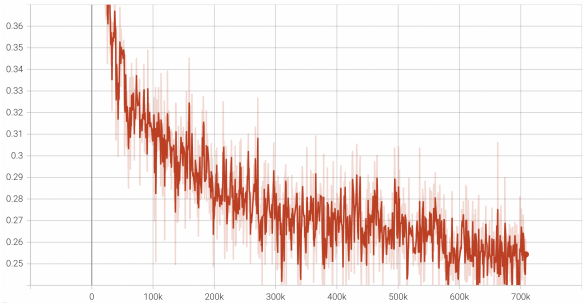

Figure 5: Proportion of tokens with rank label -1 in the masked positions for PEER-Base during the pre-training.

B Pre-training Details

The following pre-training details apply to our PEER and its competing methods including the BERT, the ELECTRA and the TEAMS. We always use Adam as the optimizer with weight decay. We mostly use the same hyperparameters as BERT and ELECTRA. Our own implementation does not include the next sentence prediction (NSP) task proposed in the original BERT, as the recent works such as Liu et al. (2019) have suggested that it does not improve the performance. We searched for the best learning rate for Small models out of [1e-3, 7.5e-4, 5e-4] . Otherwise, we did no hyperparameter tuning beyond the experiments. The full set of hyperparameters is listed in Table 8.

K	$\{\tau_1, \cdots, \tau_{K-2}\}$	G-AVG Mean±Std	MNLI	CoLA	SST-2	MRPC	STS-B	QQP	QNLI	RTE
3	$\{1\}$	81.40±0.25	81.08	59.70	89.40	87.99	87.70	90.12	88.45	66.71
4	$\{1, 10\}$	81.13±0.14	81.06	59.08	88.90	87.45	87.59	90.00	88.39	66.57
5	$\{1, 8, 32\}$	81.28±0.12	80.77	59.68	88.74	87.79	87.45	90.08	88.15	67.58

Table 5: Comparison of PEER-Small models with different K levels (under 212.5K pre-training steps) on the GLUE dev set.

δ	G-AVG Mean±Std	MNLI	CoLA	SST-2	MRPC	STS-B	QQP	QNLI	RTE
0	81.14±0.44	81.00	58.89	88.78	87.94	87.48	90.06	88.35	66.64
3	81.40±0.25	81.08	59.70	89.40	87.99	87.70	90.12	88.45	66.71
9	81.27±0.42	80.97	59.62	89.16	87.58	87.60	90.07	88.38	66.79

Table 6: Comparison of PEER-Small models with different δ values on the GLUE dev set. Each PEER-Small model has 3 levels and is pre-trained 212.5K steps.

C Fine-tuning Details

We originally fine-tuned all the pre-trained models for 4 epochs. However, because we observed a large variance in the small tasks in GLUE, following the advice from Mosbach et al. (2020), we increase the fine-tuning process to 20 epochs and select the best epoch for the four small tasks including CoLA, MRPC, STS-B and RTE. For Small models, we searched for the best learning rate out of [1e-4, 7.5e-5]. For Base models, we searched for a learning rate out of [5e-5, 3e-5] without the layer-wise learning-rate decay proposed by ELECTRA, but otherwise used the same hyperparameters as for small models. Due to limited computation resource, we adjust the number of independent fine-tuning runs (with different random seeds) so that we fine-tune more times for these tasks with smaller data sizes (i.e., with more variability). The full set of hyperparameters is listed in Table 9. Following the BERT and the ELECTRA, we do not show results on the WNLI GLUE task for the Dev set results.

C.1 Details about GLUE

We provide further details about the GLUE benchmark tasks as follows.

CoLA: Corpus of Linguistic Acceptability (Warstadt et al., 2019). The task is to determine whether a given sentence is linguistically acceptable or not. The dataset contains 8.5k train examples from books and journal articles on linguistic theory.

SST-2: Stanford Sentiment Treebank (Socher et al., 2013). The task is to determine if the sentence is positive or negative in sentiment. The dataset contains 67k train examples from movie reviews.

MRPC: Microsoft Research Paraphrase Corpus (Dolan and Brockett, 2005). The task is to predict whether two sentences are semantically equivalent or not. The dataset contains 3.7k train examples from online news sources.

STS-B: Semantic Textual Similarity (Cer et al., 2017). The task is to predict how semantically similar two sentences are on a 1-5 scale. The dataset contains 5.8k train examples drawn from news headlines, video and image captions, and natural language inference data.

QQP: Quora Question Pairs (Iyer et al., 2017). The task is to determine whether a pair of questions are semantically equivalent. The dataset contains 364k train examples from the community question-answering website Quora.

MNLI: Multi-genre Natural Language Inference (Williams et al., 2017). Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis, contradicts the hypothesis, or neither. The dataset contains 393k train examples drawn from ten different sources.

QNLI: Question Natural Language Inference; constructed from SQuAD (Rajpurkar et al., 2016). The task is to predict whether a context sentence contains the answer to a question sentence. The dataset contains 108k train examples from Wikipedia.

RTE: Recognizing Textual Entailment (Giampiccolo et al., 2007). Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis or not.

δ	G-AVG	MNLI	CoLA	SST-2	MRPC	STS-B	QQP	QNLI	RTE
0	86.42	86.72	66.44	92.09	90.20	90.75	91.69	92.59	80.87
3	86.63	86.88	68.24	92.43	90.20	90.63	91.70	92.48	80.51
9	86.77	86.69	68.57	92.66	91.18	90.92	91.78	92.57	79.78

Table 7: Comparison of PEER-Base models with different δ values on the GLUE dev set. Each PEER-Base model has 3 levels and is pre-trained 666.9K steps.

Hyperparameter	ELECTRA-Small	All Other Small Models	All Base Models
Number of layers	12	12	12
Hidden size	256	256	768
FFN inner hidden size	1024	1024	3072
Attention heads	4	4	12
Attention head size	64	64	64
Embedding size	128	256	768
Sequence length	128	128	512
Mask percent	15	15	15
Learning rate decay	Linear	Linear	Linear
Warmup steps	10000	10000	10000
Learning rate	1e-3	1e-3/7.5e-4	2e-4
Adam ϵ	1e-6	1e-6	1e-6
Adam β_1	0.9	0.9	0.9
Adam β_2	0.999	0.999	0.999
Attention dropout	0.1	0.1	0.1
Dropout	0.1	0.1	0.1
Weight decay	0.01	0.01	0.01
Batch size	512	512	256
Train steps	250K	250K	666.9K - 1M

Table 8: Pre-training hyperparameters for all the models pre-trained by us.

The dataset contains 2.5k train examples from a series of annual textual entailment challenges.

Hyperparameter	Value
Learning rate	1e-4, 7.5e-5 for Small; 5e-5, 3e-5 for Base
Adam ϵ	1e-6
Adam β_1, β_2	0.9, 0.999
Layerwise LR decay	None
Learning rate decay	Linear
Warmup fraction	0.1
Attention dropout	0.1
Dropout	0.1
Weight decay	None
Batch size	16, 32
Train epochs	20 for CoLA, MRPC, STS-B, RTE; 4 for other tasks
Seeds	5 for CoLA, MRPC, STS-B, RTE; 3 for QNLI, SST2; 1 for MNLI, QQP

Table 9: Fine-tuning hyperparameters for all the investigated models.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Section 5, and Section Limitations

☒ A2. Did you discuss any potential risks of your work?
My work has no particular risk other than the well-known risks existing in a general pre-training method.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☒ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
No response.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
No response.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
No response.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
No response.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
No response.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
No response.

C ☑ Did you run computational experiments?

4

☐ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
No response.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
4

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
4

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
4

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.