# SAS: Self-Augmentation Strategy for Language Model Pre-training

**Yifei Xu[1*], Jingqiao Zhang[2*], Ru He[2*], Liangzhu Ge[2*], Chao Yang[2], Cheng Yang[2†], Ying Nian Wu[1]**

[1] University of California, Los Angeles [2] Alibaba Group
fei960922@ucla.edu, {jingqiao.zhang, ru.he, liangzhu.glz,
xiuxin.yc, charis.yangc}@alibaba-inc.com, ywu@stat.ucla.edu

## Abstract

The core of self-supervised learning for pre-training language models includes pre-training task design as well as appropriate data augmentation. Most data augmentations in language model pre-training are context-independent. A seminal contextualized augmentation was recently proposed in ELECTRA (Clark et al. 2020b) and achieved state-of-the-art performance by introducing an auxiliary generation network (generator) to produce contextualized data augmentation for the training of a main discrimination network (discriminator). This design, however, introduces extra computation cost of the generator and a need to adjust the relative capability between the generator and the discriminator. In this paper, we propose a self-augmentation strategy (SAS) where a single network is utilized for both regular pre-training and contextualized data augmentation for the training in later epochs. Essentially, this strategy eliminates a separate generator and uses the single network to jointly conduct two pre-training tasks with MLM (Masked Language Modeling) and RTD (Replaced Token Detection) heads. It avoids the challenge to search for an appropriate size of the generator, which is critical to the performance as evidenced in ELECTRA and its subsequent variant models. In addition, SAS is a general strategy that can be seamlessly combined with many new techniques emerging recently or in the future, such as the disentangled attention mechanism from DeBERTa (He et al. 2020). Our experiments show that SAS is able to outperform ELECTRA and other state-of-the-art models in the GLUE tasks with similar or less computation cost.

## Introduction

In recent years, language model pre-training has achieved enormous success in various natural language processing (NLP) downstream tasks, largely due to innovations in pre-training task design as well as associated data augmentation. Especially, the BERT model (Devlin et al. 2018) is a milestone, where each input token sequence is augmented by randomly masking a proportion (typically 15%) of tokens and then a masked language model (MLM) task is undertook by recovering each masked token out of the vocabulary from the augmented input. Since its success, BERT has been refined or extended in the literature by adding or varying pre-training tasks and/or data augmentation. For example, the RoBERTa (Liu et al. 2019) empirically shows that dynamically masking tokens boosts BERT's performance, while pairing sentences and undertaking a corresponding next sentence prediction (NSP) task does not; the SpanBERT (Joshi et al. 2019) masks spans of tokens in each input sequence and then uses two boundary tokens of each masked span to predict each token in the span; the StructBERT (Wang et al. 2019) shuffles a proportion of unmasked tri-grams in each input sequence and then re-constructs the right order of these tokens; the XLNet (Yang et al. 2019) permutes the token index order in each input sequence and then autoregressively predicts the ending tokens in each given order; the UNILM (Dong et al. 2019) employs different self-attention masks to augment input data in three ways so that unidirectional, bidirectional and sequence-to-sequence LM tasks can be performed together; the ERNIE (Sun et al. 2019) uses prior knowledge to enhance its data augmentation by phrase-level masking and entity-level masking; the CLEAR (Wu et al. 2020), while keeping the MLM task, generates a pair of augmented sentences from each input sentence using four augmentation methods (words deletion, spans deletion, synonym substitution and reordering) and then performs a sentence-level contrastive learning task to minimize the distance between augmented sentences coming from the same original sentence. In all these models, data augmentations are manually designed and context-independent without taking sequence context into account.

Recently, the ELECTRA (Clark et al. 2020b) model is proposed to construct contextualized data augmentation. It trains an auxiliary generation network to provide plausible tokens to replace a proportion (typically 15%) of the original tokens according to the input context, and then utilize a main discrimination network to classify whether each token in the augmented input is a replaced token or not. The latter task was named as replaced token detection (RTD). After pre-training, the generator is discarded and the discriminator is further fine-tuned for downstream NLP tasks. ELECTRA has shown impressive advantages over MLM-based models in various downstream tasks under similar computation cost, especially when a model size is small. Since the ELECTRA model needs to train two separate neural networks (i.e., generator and discriminator) side by side, however, it is chal-

*These authors contributed equally.
†Corresponding author: Cheng Yang.

lenging to maintain an appropriate level of generator's ability relative to discriminator's during the training. When the generator becomes stronger than what the discriminator is capable of dealing with, it will prevent effective learning of the discriminator. In addition, training two separate networks naturally leads to extra computation cost.

To address the aforementioned disadvantages, we propose a self-augmentation strategy (SAS) that uses a single neural network to provide contextualized data augmentation for its own use in subsequent epochs during the training.[1] The SAS network includes two light-weight heads (MLM head and RTD head) on the top of one common transformer encoder (Vaswani et al. 2017). This network architecture allows automatic adjustment of the relative difficulty of the generated tokens from the MLM head, while jointly undertaking both MLM and RTD pre-training task. As a result, it shows better performance in downstream GLUE tasks (Wang et al. 2018) than ELECTRA, with similar computation efficiency.

The advantages of SAS can be summarized as follows:

- Compared to ELECTRA and its subsequent variants that utilize an auxiliary network to generate contextualized data augmentation, SAS introduces a single network to jointly generate contextualized data augmentation and perform pre-training tasks. On one hand, it avoids the challenge to balance the capability of the generator and that of other model components. On the other hand, it has both MLM and RTD heads on the top of the same encoder, so that the pre-trained SAS model have a foundation for both generation and discrimination capabilities which is beneficial to the fine-tuning of different downstream tasks. As a comparison, in the case of ELECTRA, the generation network is discarded and only the discrimination network is used for downstream applications.
- SAS eliminates the computation cost related to the generation network and therefore improves computation efficiency. As ELECTRA has already shown computation efficiency especially for small models, SAS provides an extra economic pre-training solution when compute resources is limited in many practices.
- SAS is a simple and general strategy and can be easily integrated with many other techniques emerging recently or in the future. Taking as an example, we have incorporated the disentangled attention mechanism proposed in DeBERTa (He et al. 2020) and further outperform the state-of-the-art models on the GLUE benchmark.

## Related Work

**Pre-trained Language Models:** As introduced in the previous section, many pre-trained language models have emerged with new performance records in many downstream NLP tasks in the last several years. Among them, ELECTRA (Clark et al. 2020b) distinguishes itself from previous models and inspires our SAS model, since it introduces contextualized data augmentation with an auxiliary generation network and then train a main discrimination network to perform the RTD task.

[1]In this paper, we focus on the field of language modeling and note that SAS is a general strategy that can also be applied to other domains such as computer vision and pattern recognition.

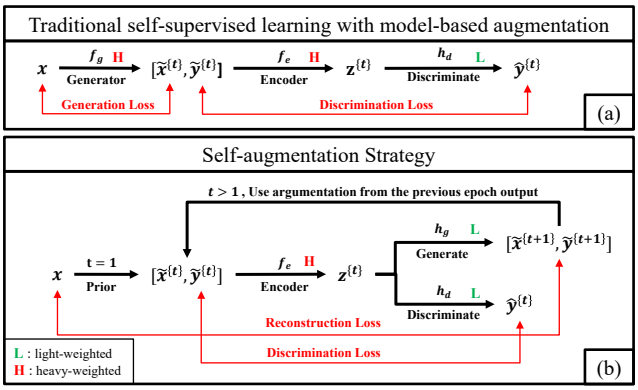


Figure 1: The comparison between the traditional self-supervised learning with model-based augmentation and the self-augmentation strategy.

The Electric model is later 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. This model also needs a separate network to provide noise distribution in order to generate contextualized data augmentation. The MC-BERT model is proposed by Xu et al. (2020b) to replace the RTD binary classification task in ELECTRA with a multi-class classification task so that richer semantic information can be captured. While MC-BERT performs on par with ELECTRA, it also needs a separate network called meta controller to generate a multi-class candidate set for each token as contextualized data augmentation.

COCO-LM (Meng et al. 2021) has been recently proposed to improve ELECTRA by using two new pre-training tasks called Corrective Language Modelling (CLM) and Sequence Contrastive Learning tasks. In particular, the CLM refines the All-Token MLM task studied by Clark et al. (2020b) and jointly learns the RTD and MLM tasks by a combination of two loss objectives at all output token positions. This is similar to SAS; however, a key caveat is that the MLM head, which is computationally much heavier than the RTD head due to its softmax calculation over the vocabulary, is only conducted at the augmented positions in SAS. COCO-LM also relies on a separate generation network to generate the augmented sequence for its two tasks which leads to extra computation. In contrast, the proposed SAS eliminates the requirement of a separate generator and achieves higher computation efficiency.

Another recent work is the DeBERTa model (He et al. 2020), which computes the attention weights among tokens using disentangled matrices on two separate vectors (content vector and relative position vector) of each token. Because SAS is a general strategy, it can seamlessly incorporate such a disentangled attention mechanism to further improve performance, as will be shown in Section of Experiments.

**Teacher-student Mechanism:** SAS can be alternatively viewed from the perspective of a teacher-student mechanism, where the knowledge of a teacher model is utilized to facilitate the learning process of a student model. In essence, in the SAS method, we treat the model trained in the current epoch as a student and that in the previous epoch as a (weak) teacher, since the latter generates contextualized data augmentation to help the former to learn.

The teacher-student mechanism has been traditionally used in knowledge distillation for the purpose of label augmentation (Hinton, Vinyals, and Dean 2015), where soft labels generated from a teacher is combined with hard (one-hot) labels observed from the original data to serve as the ultimate label to supervise the learning of a student. The size of a student could be either smaller than or the same as that of a teacher. Sanh et al. (2019) shows that using the original BERT as a teacher their DistilBERT (student) model is able to remain 97% of the teacher's performance in the downstream GLUE tasks, while the size of the student is only 60% of that of the teacher. When a student has the same size as its teacher during the knowledge distillation, the advantages of student over teacher have been empirically shown in recent research. For example, Kim et al. (2020) show the benefits of their Teacher-free Knowledge Distillation method by using a pre-trained first-generation model as a teacher, while Yuan et al. (2020) show the gains of their Self-knowledge Distillation method by using a (student) model itself in the previous epoch during the training as a teacher.

In recent years, the teacher-student mechanism has also been used to obtain state-of-the-art results in semi-supervised settings. Laine and Aila (2016) propose Temporal Ensembling method to use an EMA (exponential moving average) of the output predictions of the model in the previous epochs as a teacher prediction to constitute the consistency loss component for each data instance, in addition to the cross-entropy loss component for each labeled data instance. Tarvainen and Valpola (2017) propose to use an EMA of parameter values (i.e., weights) of (student) model to construct a mean teacher for the consistency loss component for each data instance.

**Curriculum Learning and Self-paced Learning** Curriculum learning and self-paced learning have attracted increasing attention in machine learning. Both of them generally claim that it helps the learning process to start learning with easier data and then gradually take more complex or difficult data into consideration. There are some notable works that intend to use them in various NLP tasks (Wan et al. 2020; Xu et al. 2020a; Nijkamp et al. 2021). In SAS, we naturally provides a curriculum for the model itself by incrementally augmenting the data as MLM improves generation capability over the training process. In essence, our model can be regarded as a combination of the two learning strategies which adaptively generate augmented data for better learning. This kind of idea dates back to the work of Jiang et al. (2015), which is a pioneer of combining these two strategies.

# Method

In this section, we formulate the self-augmentation strategy as a general framework that can be applied to various domains, and then detail its usage in language model pre-training.

## General SAS Framework

In supervised learning, each data instance has both feature $\boldsymbol{x}_i$ and its corresponding label $\boldsymbol{y}_i$. For example, in the language model pre-training, since there is no label in the training dataset, an input pair of $\tilde{\boldsymbol{x}}_i$ and label $\tilde{\boldsymbol{y}}_i$ is constructed by an augmentation process $q$ from the original data instance of token sequence $\boldsymbol{x}_i \sim \mathcal{X}$. The augmentation process can be based on an independent random process (such as random masking, random token replacement with a certain distribution, etc.) or a separate generation network $f_g$ with parameter $\theta_g$ to consider sequence content. Let's consider the latter and denote the augmented dataset as $\tilde{\mathcal{X}} = \{[\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{y}}_i] = f_g(\boldsymbol{x}_i) | \boldsymbol{x}_i \sim \mathcal{X}; i \in \{1, \cdots, n\}\}$. As shown in Figure 1(a), the augmented data instance is fed to a heavy-weight main network with encoder $f_e$ with parameter $\theta_e$ and a light-weight head $h_d$ with parameter $\theta_d$ to conduct pre-training task. As an example, a discrimination head RTD is applied in ELECTRA to detect whether each input token has been replaced/augmented or not. In a forward pass, the auxiliary and main networks are used sequentially. The computation cost of the auxiliary generator $f_g$ is usually not ignoble (e.g., it is recommended to be 1/4 to 1/2 of the size of the main discrimination network after extensive experiments in ELECTRA.)

In SAS, we eliminate the separate generator in augmentation. Instead, we introduce both generation and discrimination capabilities via a single network with both MLM and RTD heads, $h_g$ and $h_d$, and design a light-weight generation process by using the softmax output of MLM. That is, the MLM head serves dual purposes for both pre-training and contextualized data augmentation. In the first epoch ($t = 1$), SAS experiences a cold start, since there is no prior MLM output yet. To start with, a simple prior random distribution $p^{\{0\}}$ is applied. In the $(t+1)$'th epoch for $t \geq 1$, augmented data is generated based on MLM's softmax output of the previous epoch. During the training, while $h_g$ is updated from generation loss and $h_d$ from discrimination loss, the encoder $f_e$ is updated from both losses.

## SAS in Language Model Pre-training

Now we describe the details of SAS in the context of language model pre-training. As mentioned above, SAS has a single neural network that includes two light-weight heads on the top of one common heavy-weight transformer encoder. It undertakes both the MLM and RTD tasks in each forward pass during the training process. A workflow of SAS is demonstrated in Figure 2.

For an instance $\boldsymbol{x} = (x_1, x_2, \cdots, x_k)$ of a token sequence of length $k$ in the pre-training dataset $\mathcal{X}$, in the $t$'th epoch of the pre-training process, we choose a random set $S^{\{t\}}$ with $\lceil 0.15 \cdot k \rceil$ position indexes, just as BERT and ELECTRA, and create augmented input instance $\tilde{\boldsymbol{x}}^{\{t\}} =$

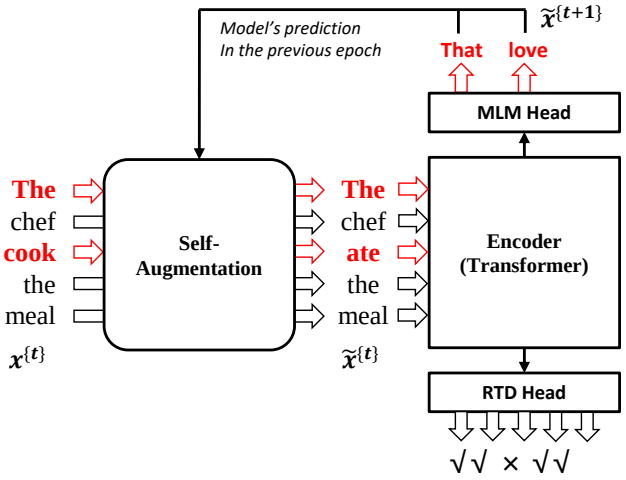


Figure 2: Workflow of SAS in language model pre-training.

$(\tilde{x}_1^{\{t\}}, \tilde{x}_2^{\{t\}}, \cdots, \tilde{x}_k^{\{t\}})$ as follows,

$$\begin{cases} \tilde{x}_i^{\{t\}} = x_i & i \notin S^{\{t\}} \\ \tilde{x}_i^{\{t\}} \sim p^{\{t-1\}} & i \in S^{\{t\}} \end{cases} \tag{1}$$

In other words, we keep original token $x_i$ if $i \notin S^{\{t\}}$ and otherwise replace it by sampling a token based on a generating distribution $p^{\{t-1\}}$ to be detailed below.

In the first epoch ($t = 1$), we set $p^{\{0\}}$ as a cold-start token generating distribution, such as a (token-frequency based) unigram distribution. Thereafter, in epoch $t + 1$ ($t \geq 1$), we leverage the softmax output $p^{\{t\}}_{\theta_e,\theta_g}(x_i|\tilde{\boldsymbol{x}}^{\{t\}})$ of the MLM head in the $t$'th epoch as an enhanced contextualized token generating distribution. We have

$$p^{\{t\}}_{\theta_e,\theta_g}(x_i|\tilde{\boldsymbol{x}}^{\{t\}}) = \frac{\exp\{e(x_i)^T h_g(f_e(\tilde{\boldsymbol{x}}^{\{t\}}))_i\}}{\sum_{x' \in V} \exp\{e(x')^T h_g(f_e(\tilde{\boldsymbol{x}}^{\{t\}}))_i\}}, \tag{2}$$

where $e(x_i)$ is the embedding of token $x_i$, $f_e$ is the encoder that outputs a sequence of $k$ contextualized representation vectors from the augmented input $\tilde{\boldsymbol{x}}$, $h_g$ is the MLM head that generates representation vector at each augmented position in $S^{\{t\}}$, and $V$ is the vocabulary. Essentially, the SAS model itself in the $t$'th epoch serves as a teacher to provide data augmentation for its own learning in the $(t+1)$'th epoch.

In practice, with the distribution $p^{\{t\}}_{\theta_e,\theta_g}(x_i|\tilde{\boldsymbol{x}}^{\{t\}})$ generated from the MLM head in each forward pass in the $t$'th epoch, we directly sample $\tilde{x}_i^{\{t+1\}}$ from it for every $i \in S^{\{t+1\}}$ and then store the sampled token index in CPU memory for the augmentation usage in the $(t+1)$'th epoch. This way, we avoid storing the whole distribution with high memory cost. The loss of the MLM task is calculated as a softmax loss,

$$\mathcal{L}^{\{t\}}_{mlm}(\boldsymbol{x}; \theta_e, \theta_g) = -\mathbb{E}\left[\sum_{i \in S^{\{t\}}} \log p^{\{t\}}_{\theta_e,\theta_g}(x_i|\tilde{\boldsymbol{x}}^{\{t\}})\right]. \tag{3}$$

We adopt the same RTD task as ELECTRA to classify whether each token in the augmented input $\tilde{\boldsymbol{x}}^{\{t\}}$ is a replaced token or not. The corresponding classification probability is

$$D(\tilde{\boldsymbol{x}}^{\{t\}})_i = \sigma(h_d(f_e(\tilde{\boldsymbol{x}}^{\{t\}}))_i), \tag{4}$$

where $\sigma$ is the logistic sigmoid, $h_d$ is the RTD head function that generates a sequence of logits at all the positions. Note that the same $f_e$ is used in Eq. (2) and Eq. (4), indicating that only one forward pass through the encoder is needed. This largely reduces the computation cost, in contrast to ELECTRA's separate forward passes in two neutral networks.

Eq. (4) leads to the loss of the RTD task given the instance $\boldsymbol{x}$ in the $t$'th epoch as

$$\begin{aligned} \mathcal{L}^{\{t\}}_{rtd}(\boldsymbol{x}; \theta_e, \theta_d) = -\mathbb{E}\Bigg[\sum_{j=1}^{k} y_j^{\{t\}} \log D(\tilde{\boldsymbol{x}}^{\{t\}})_j \\ + (1 - y_j^{\{t\}}) \log(1 - D(\tilde{\boldsymbol{x}}^{\{t\}})_j)\Bigg], \end{aligned} \tag{5}$$

where $y_j^{\{t\}} = 1$ if $\tilde{x}_j^{\{t\}} = x_j$, $y_j^{\{t\}} = 0$ otherwise.

We train the SAS model by minimizing a combined loss:

$$\sum_{\boldsymbol{x} \in \mathcal{X}} \mathcal{L}^{\{t\}}_{mlm}(\boldsymbol{x}; \theta_e, \theta_g) + \lambda^{\{t\}} \mathcal{L}^{\{t\}}_{rtd}(\boldsymbol{x}; \theta_e, \theta_d), \tag{6}$$

where $\lambda^{\{t\}}$ is the (relative) weight of the RTD loss which can be a constant or varied across different epochs. The joint-task learning with the combines loss, along with the shared encoder for both heads, avoids learning obstacles from potentially over-challenging replaced tokens.

## Experiments

In this section, we provide experimental details and comprehensive study of results.

### Experimental Setup

**Pre-training Details:** Our implementation[2] is based on Huggingface Transformers 4.3 framework (Wolf et al. 2020). We include ELECTRA, DeBERTa as well as BERT for comparison. Under 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 aligned with ELECTRA. 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 (FLOPs is a measure agnostic to the particular hardware and low-level optimizations). We reuse the FLOPs

[2]Code and pretrained model is available publicly at Github: https://github.com/fei960922/SAS-Self-Augmentation-Strategy.

computation code[3] 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 train ELECTRA-Base and ELECTRA-Small 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 0.25M pre-training steps, instead of batch size 128 and 1M steps in Clark et al. (2020b), and double the learning rate accordingly.[4] 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).
- **DeBERTa**: We implement DeBERTa and adopt its idea of sharing the projection matrices between relative position embedding and content embedding in all the attention layers proposed by He et al. (2020). Such a sharing does retain the model performance on the small and base models while reducing the number of model parameters.
- **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 hidden size[5], according to the convention of the BERT models. Our BERT-Small setting makes its FLOPs similar to that of ELECTRA-Small when the training steps are the same, so that a fair comparison of their performance be made directly.
- **SAS**: We experiment three SAS settings:
  - SAS denotes the setting with a default weight scheduler that epoch-wisely increases the RTD loss weight $\lambda^{\{t\}}$ from 50 to 200 during the pre-training.
  - SAS$^{c}$ denotes a setting where RTD loss weight $\lambda^{\{t\}}$ is a constant value 50.
  - SAS$^{\text{DA}}$ denotes a setting that uses the default weight scheduler and further incorporates the disentangled attention mechanism.

In all the SAS settings, we set the (token-frequency based) unigram distribution as the cold-start distribution.

**Downstream Tasks:** 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, Nangia, and Bowman 2017), RTE (Giampiccolo et al. 2007), and QNLI (Rajpurkar et al. 2016); semantic similarity tasks MRPC (Dolan and Brockett 2005), QQP (Iyer, Dandekar, and Csernai 2017), and STS-B (Cer et al. 2017); sentiment classification task SST-2 (Socher et al. 2013); and linguistic acceptability classification CoLA (Warstadt, Singh, and Bowman 2019). See supplementary for more details on the GLUE tasks.

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

**Fine-tuning Procedure:** For the fine-tuning of GLUE, 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 (such as 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, Andriushchenko, and Klakow 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 the supplementary for detailed 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 runs.

## Overall Comparison Results

The performance comparison among the small models is shown in Table 1. In the table, the second column shows the training FLOPs of each model, and the third column lists the mean and the standard deviation of GLUE Score for each model across five independently pre-trained checkpoints.

As for the three competing models, it shows that both ELECTRA-Small and DeBERTa-Small outperform BERT-Small in terms of GLUE Score. Even when we increase the pre-training steps of BERT-Small model from 250K to 375K, denoted by BERT-Small-1.5x in the table, it still is worse than ELECTRA-Small while being similar to DeBERTa-Small. The GLUE Score of ELECTRA-Small (our implementation) is about 98.45% of that of ELECTRA-Small++ released by Clark et al. (2020b) (80.78 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. Note that DeBERTa-Small has much higher MNLI mean score than ELECTRA-Small, while ELECTRA-Small has higher mean of GLUE Score which is aligned with the overall advantage of ELECTRA in small models emphasized by Clark et al. (2020b).

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

[4] We observe that such a change is able to significantly reduce the pre-training time without degrading the model performance.

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

| Model | Train FLOPs | GLUE mean±std | CoLA | SST-2 | MRPC | STS-B | QQP | MNLI | QNLI | RTE |
|---|---|---|---|---|---|---|---|---|---|---|
| BERT-Small | 1.274e18 | 79.10±0.07 | 49.69 | 90.14 | 84.64 | 86.04 | 89.51 | 80.00 | 86.84 | 65.94 |
| BERT-Small-1.5x | 1.911e18 | 79.55±0.09 | 51.80 | 90.37 | 84.07 | 86.04 | 89.80 | 80.20 | 86.58 | 67.51 |
| ELECTRA-Small | 1.294e18 | 80.78±0.18 | 59.27 | 89.18 | 86.40 | 86.76 | 89.92 | 80.29 | 88.23 | 66.21 |
| *ELECTRA-Small++** | *2.403e19* | *82.05* | *58.37* | *91.40* | *87.01* | *87.95* | *90.54* | *82.52* | *88.93* | *69.68* |
| DeBERTa-Small | 1.381e18 | 79.52±0.43 | 49.51 | 89.91 | 86.68 | 86.29 | 90.26 | 81.51 | 87.78 | 64.26 |
| SAS$^c$-Small | 1.279e18 | 81.30±0.13 | 59.52 | 89.60 | 87.17 | 87.27 | 90.20 | 81.54 | 88.67 | 66.43 |
| SAS-Small | 1.279e18 | 81.61±0.24 | 60.49 | 90.08 | 87.01 | 87.32 | 90.11 | 81.37 | 88.47 | **68.05** |
| SAS$^{DA}$-Small | 1.385e18 | **82.14±0.22** | **62.35** | **90.55** | **87.55** | **87.52** | **90.60** | **82.20** | **88.71** | 67.65 |

*: ELECTRA-Small++ is the pre-trained model publicly released by Clark et al. (2020b). It uses 18x training FLOPs compared to ELECTRA-Small.

Table 1: Comparison of small models on the GLUE dev set.

| Model | Train FLOPs | GLUE | CoLA | SST-2 | MRPC | STS-B | QQP | MNLI | QNLI | RTE |
|---|---|---|---|---|---|---|---|---|---|---|
| *BERT-Base (1M)** | *6.430e19* | *83.06* | *60.07* | *92.09* | *85.29* | *89.22* | *91.27* | *83.99* | *91.43* | *71.12* |
| ELECTRA-Base (766K) | 6.426e19 | 85.46 | 65.53 | 91.28 | **89.95** | **90.33** | 91.65 | 85.49 | **91.85** | **77.62** |
| DeBERTa-Base (1M) | 7.443e19 | 83.97 | 58.46 | 93.23 | 88.97 | 89.36 | 91.37 | 85.53 | 91.52 | 73.29 |
| SAS$^{DA}$-Base (833K) | **6.226e19** | **85.60** | **66.56** | **93.35** | 88.73 | 90.05 | **91.73** | **86.49** | 91.74 | 76.17 |

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

Table 2: Comparison of base models on the GLUE dev set.

The proposed SAS model further improves the overall performance over ELECTRA and DeBERTa with fewer FLOPs.[6] The mean of GLUE Score of SAS$^c$-Small (with the constant weight for the RTD loss) is 0.52 point higher than that of ELECTRA-Small, while SAS with the epoch-wise weight increasing strategy is 0.83 point higher. As for MNLI task, both SAS$^c$-Small and SAS have their mean scores more than 1 point higher than ELECTRA.

Notably, SAS$^{DA}$-Small, the SAS with the disentangled attention mechanism, achieves a higher GLUE Score than ELECTRA-Small++'s with only 5.76% of FLOPs consumed by ELECTRA-Small++. The disentangled attention mechanism embraced by SAS is able to improve the mean of GLUE Score by 0.65% (from 81.61 to 82.14). This is larger than its 0.53% improvement ratio on the basis of BERT (from 79.10 to 79.52), which shows that SAS can effectively realize the value of the disentangled attention mechanism. The table also shows that DeBERTa-Small has the largest standard deviation among all the models. This might be because DeBERTa-Small needs larger data (such as the data used by He et al. (2020)) to stably achieve its functionality.

The comparison results on the base models are shown in Table 2. In the first column of the table, we show the pre-training steps of each model and have ensured that SAS takes similar or fewer FLOPs than other models. The table shows that SAS$^{DA}$-Base[7] is able to achieve both the highest GLUE Score (ELECTRA-Base is a very close second) and the highest MNLI score (nearly 1 point higher than the second best DeBERTa-Base).

[6]With 1 V100 GPU, the pre-training of SAS$^{DA}$-Small takes 37.5h; both SAS-Small and SAS$^c$-Small takes about 24h; and ELECTRA-Small takes about 35h.

[7]The pre-training costs 7.7 days by 8 V100 GPUs.

## Ablation Study

We design the following variants to investigate the contributions of different components of SAS:

- Unig-MLM denotes a variant of BERT with random replacement of 15% of tokens based on the (token-frequency based) unigram distribution. That is, each of the randomly selected tokens is replaced with a token sampled from the unigram distribution instead of a [MASK] token.
- Unig-MLM-SAS denotes a variant of SAS with only MLM task instead of two combined tasks.
- Unig-MLM-RTD$^c$ denotes a model that is the same as SAS$^c$ but use a random replacement with the unigram distribution in the whole training process.
- Unig-MLM-RTD denotes a model that is the same as SAS but use a random replacement with the unigram distribution in the whole training process.

Table 3 summarizes the results on the small models. First, it shows slight drop (0.1 point) in GLUE Score if the random masking of BERT is changed to the random replacement, which might be among the reasons that BERT uses the special [MASK] token despite its mismatch with the fine-tuning data. Second, when we use SAS to conduct the MLM task alone, it only increases the mean of GLUE Score from 79.10 to 79.38 compared to BERT, which indicates that the MLM task cannot fully utilize the self-augmented data to learn a better model. Third, by comparing Unig-MLM to Unig-MLM-RTD (or Unig-MLM-RTD$^c$), we observe that a big (about 1.4 point) increase in the mean of GLUE Score comes from adding the RTD task with the MLM task. This increase is mainly due to the improvement in CoLA, as it alone contributes more than a 1.2 point increase in the mean of GLUE Score. This indicates that the RTD task along with the data

| Model | GLUE mean±std | CoLA | SST-2 | MRPC | STS-B | QQP | MNLI | QNLI | RTE |
|---|---|---|---|---|---|---|---|---|---|
| Mask-MLM (BERT) | 79.10 ±0.12 | 49.69 | 90.14 | 84.64 | 86.04 | 89.51 | 80.00 | 86.84 | 65.94 |
| Unig-MLM | 79.00 ±0.07 | 49.61 | 90.06 | 84.97 | 85.43 | 89.42 | 80.11 | 86.93 | 65.46 |
| Unig-MLM-SAS | 79.38 ±0.36 | 49.94 | 89.72 | 86.13 | 85.48 | 89.69 | 80.34 | 87.43 | 66.28 |
| Unig-MLM-RTD$^c$ | 80.43 ±0.14 | 59.28 | 89.37 | 85.46 | 85.47 | 89.87 | 80.50 | 88.17 | 65.34 |
| Unig-MLM-RTD | 80.40 ±0.15 | 59.96 | 89.53 | 85.05 | 85.36 | 89.76 | 80.56 | 87.97 | 64.98 |
| SAS$^c$ | 81.30 ±0.13 | 59.52 | 89.60 | 87.17 | 87.27 | 90.20 | 81.54 | 88.67 | 66.43 |
| SAS | 81.61 ±0.24 | 60.49 | 90.08 | 87.01 | 87.32 | 90.11 | 81.37 | 88.47 | 68.05 |

Table 3: Ablation study on small models on the GLUE dev set.

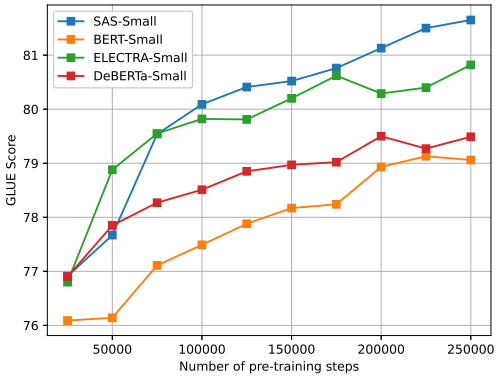


Figure 3: Curve of GLUE Score for SAS-Small model and its competitors with respect to the pre-training steps on the GLUE dev set.

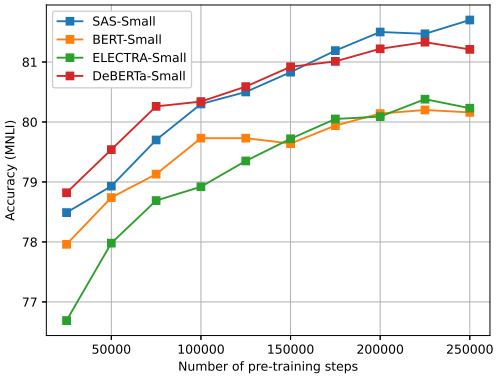


Figure 4: Curve of MNLI's average accuracy for SAS-Small model and its competitors with respect to the pre-training steps on the GLUE dev set.

augmented from the unigram distribution can greatly help the syntactic task (CoLA) but not other semantic tasks in GLUE. Finally, by comparing Unig-MLM-RTD (or Unig-MLM-RTD$^c$) to SAS (or SAS$^c$), we see another notable increase in the mean of GLUE Score, which comes from the improvement in both the syntactic task (CoLA) and other semantic tasks such as MNLI. This shows that SAS can further improve a variety of different tasks, as its pre-trained model has both generation and discrimination capabilities with MLM and RTD heads together. The effect of the weight strategy for the RTD loss can also be seen from Table 3. By comparing SAS$^c$ to SAS, we see that CoLA's score has a further increase when the epoch-wise weight increasing strategy gradually puts more emphasis on the RTD task.

### Pre-Training Efficiency

To investigate the convergence of pre-training, in Figures 3 and 4, we plot GLUE Score and MNLI score with respect to the number of pre-training steps for SAS-Small and other models. For each model, we select the median run whose pre-training random seed achieves the median GLUE Score among the five random seeds. Then for the selected median run of each model, we save a checkpoint every 25K pre-training steps (i.e., 0.5 epoch), and fine-tune it on each GLUE task and finally report GLUE Score across the tasks. In Figure 3, it shows that SAS-Small is significantly better than BERT-Small in its first (cold-start) epoch, and starts to outperform all other models after the 1.5-th epoch. Figure 4 shows that SAS-Small and DeBERTa-Small perform consistently better than BERT-Small and ELECTRA-Small for MNLI task, while SAS-Small performs on a par with DeBERTa-Small after the second epoch.

## Conclusion

We propose the self-augmentation strategy (SAS) motivated by ELECTRA's contextualized data augmentation for language model pre-trainings. SAS eliminates a separate generation network used in ELECTRA and uses a single network to generate the contextualized data augmentation for its own subsequent training. By this way, it avoids the challenge in controlling the relative capability of the generator and reduces the computation cost. In addition, our SAS is a general and open framework so that it can easily incorporate many other techniques emerging recently or in the future to further improve its performance. The experimental results show that SAS is able to outperform the previous state-of-the-art language pre-training models in the small and base scales with similar or less computation cost. We will continue to validate SAS's advantage in larger models when sufficient computation resources are available, while encouraging more studies on economic and compute-efficient pre-training approaches.

## References


Cer, D.; Diab, M.; Agirre, E.; Lopez-Gazpio, I.; and Specia, L. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*.

Clark, K.; Luong, M.-T.; Le, Q.; and Manning, C. D. 2020a. Pre-Training Transformers as Energy-Based Cloze Models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 285–294. Online: Association for Computational Linguistics.

Clark, K.; Luong, M.-T.; Le, Q. V.; and Manning, C. D. 2020b. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dodge, J.; Ilharco, G.; Schwartz, R.; Farhadi, A.; Hajishirzi, H.; and Smith, N. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*.

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

Dong, L.; Yang, N.; Wang, W.; Wei, F.; Liu, X.; Wang, Y.; Gao, J.; Zhou, M.; and Hon, H. 2019. Unified Language Model Pre-training for Natural Language Understanding and Generation. *CoRR*, abs/1905.03197.

Giampiccolo, D.; Magnini, B.; Dagan, I.; and Dolan, W. B. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, 1–9.

Gutmann, M. U.; and Hyvärinen, A. 2012. Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics. *Journal of Machine Learning Research*, 13(11): 307–361.

He, P.; Liu, X.; Gao, J.; and Chen, W. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop*.

Iyer, S.; Dandekar, N.; and Csernai, K. 2017. First quora dataset release: Question pairs. *data. quora. com.*

Jiang, L.; Meng, D.; Zhao, Q.; Shan, S.; and Hauptmann, A. 2015. Self-paced curriculum learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

Joshi, M.; Chen, D.; Liu, Y.; Weld, D. S.; Zettlemoyer, L.; and Levy, O. 2019. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *CoRR*, abs/1907.10529.

Kim, K.; Ji, B.; Yoon, D.; and Hwang, S. 2020. Self-Knowledge Distillation: A Simple Way for Better Generalization. *CoRR*, abs/2006.12000.

Laine, S.; and Aila, T. 2016. Temporal Ensembling for Semi-Supervised Learning. *CoRR*, abs/1610.02242.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, abs/1907.11692.

Meng, Y.; Xiong, C.; Bajaj, P.; Tiwary, S.; Bennett, P.; Han, J.; and Song, X. 2021. Coco-lm: Correcting and contrasting text sequences for language model pretraining. *arXiv preprint arXiv:2102.08473*.

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

Nijkamp, E.; Pang, B.; Wu, Y. N.; and Xiong, C. 2021. SCRIPT: Self-Critic Pretraining of Transformers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 5196–5202.

Rajpurkar, P.; Zhang, J.; Lopyrev, K.; and Liang, P. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Sanh, V.; Debut, L.; Chaumond, J.; and Wolf, T. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A. Y.; and Potts, C. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, 1631–1642.

Sun, Y.; Wang, S.; Li, Y.; Feng, S.; Chen, X.; Zhang, H.; Tian, X.; Zhu, D.; Tian, H.; and Wu, H. 2019. ERNIE: Enhanced Representation through Knowledge Integration. *CoRR*, abs/1904.09223.

Tarvainen, A.; and Valpola, H. 2017. Mean Teachers Are Better Role Models: Weight-Averaged Consistency Targets Improve Semi-Supervised Deep Learning Results. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 1195–1204. ISBN 9781510860964.

Taylor, W. L. 1953. "Cloze Procedure": A New Tool for Measuring Readability. *Journalism Quarterly*, 30(4): 415–433.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30, 6000–6010.

Wan, Y.; Yang, B.; Wong, D. F.; Zhou, Y.; Chao, L. S.; Zhang, H.; and Chen, B. 2020. Self-Paced Learning for Neural Machine Translation. *arXiv preprint arXiv:2010.04505*.

Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; and Bowman, S. R. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *CoRR*, abs/1804.07461.

Wang, W.; Bi, B.; Yan, M.; Wu, C.; Bao, Z.; Peng, L.; and Si, L. 2019. StructBERT: Incorporating Language Structures into Pre-training for Deep Language Understanding. *CoRR*, abs/1908.04577.

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

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

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; Davison, J.; Shleifer, S.; von Platen, P.; Ma, C.; Jernite, Y.; Plu, J.; Xu, C.; Scao, T. L.; Gugger, S.; Drame, M.; Lhoest, Q.; and Rush, A. M. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. Online: Association for Computational Linguistics.

Wu, Z.; Wang, S.; Gu, J.; Khabsa, M.; Sun, F.; and Ma, H. 2020. CLEAR: Contrastive Learning for Sentence Representation. *arXiv preprint arXiv:2012.15466*.

Xu, B.; Zhang, L.; Mao, Z.; Wang, Q.; Xie, H.; and Zhang, Y. 2020a. Curriculum learning for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 6095–6104.

Xu, Z.; Gong, L.; Ke, G.; He, D.; Zheng, S.; Wang, L.; Bian, J.; and Liu, T. 2020b. MC-BERT: Efficient Language Pre-Training via a Meta Controller. *CoRR*, abs/2006.05744.

Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J. G.; Salakhutdinov, R.; and Le, Q. V. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *CoRR*, abs/1906.08237.

Yuan, L.; Tay, F. E.; Li, G.; Wang, T.; and Feng, J. 2020. Revisiting Knowledge Distillation via Label Smoothing Regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3903–3911.

Zhang, T.; Wu, F.; Katiyar, A.; Weinberger, K. Q.; and Artzi, Y. 2020. Revisiting few-sample BERT fine-tuning. *arXiv preprint arXiv:2006.05987*.