Training ELECTRA Augmented with Multi-word Selection

Jiaming Shen$^{\nabla\star}$, **Jialu Liu**$^{\diamond}$, **Tianqi Liu**$^{\diamond}$, **Cong Yu**$^{\diamond}$, **Jiawei Han**$^{\nabla}$

$^{\nabla}$University of Illinois Urbana-Champaign, IL, USA, $^{\diamond}$Google Research, NY, USA

$^{\nabla}${js2, hanj}@illinois.edu $^{\diamond}${jialu, tianqiliu, congyu}@google.com

Abstract

Pre-trained text encoders such as BERT and its variants have recently achieved state-of-the-art performances on many NLP tasks. While being effective, these pre-training methods typically demand massive computation resources. To accelerate pre-training, ELECTRA trains a discriminator that predicts whether each input token is replaced by a generator. However, this new task, as a binary classification, is less semantically informative. In this study, we present a new text encoder pre-training method that improves ELECTRA based on multi-task learning. Specifically, we train the discriminator to simultaneously detect replaced tokens and select original tokens from candidate sets. We further develop two techniques to effectively combine all pre-training tasks: (1) using attention-based networks for task-specific heads, and (2) sharing bottom layers of the generator and the discriminator. Extensive experiments on GLUE and SQuAD datasets demonstrate both the effectiveness and the efficiency of our proposed method.

1 Introduction

Contextualized representations from pre-trained text encoders have shown great power for improving many NLP tasks (Rajpurkar et al., 2016; Wang et al., 2019b,a; Liu and Lapata, 2019). Most pre-trained encoders, despite their variety, follow BERT (Devlin et al., 2019) and adopt the masked language modeling (MLM) pre-training task which trains the model to recover the identities of a small subset of masked tokens. Although being more effective than conventional left-to-right language model pre-training (Peters et al., 2018; Radford et al., 2018) due to capturing bidirectional information, MLM-based approaches (Liu et al., 2019b; Joshi et al., 2019) can only learn from those masked

---

*This work is done while interning at Google Research. Corresponding Author: Jialu Liu.

tokens which are typically just 15% of all tokens in the input sentences.

To address the low sample efficiency issue, ELECTRA (Clark et al., 2020a) proposes a new pre-training task. Specifically, it corrupts a sentence by replacing some tokens with plausible alternatives sampled from a generator and trains a discriminator to predict whether each token in the corrupted sentence is replaced or not. After pre-training ends, it throws away the generator and exports the discriminator for down-stream applications. As the discriminator can learn from all input tokens, ELECTRA is more sample efficient than previous MLM-based methods. However, follow-up studies (Xu et al., 2020; Aroca-Ouellette and Rudzicz, 2020) find this new replaced token detection task, as a binary classification, is often too simple to learn. As a result, the discriminator output representations are insufficiently trained and encode inadequate semantic information.

In this work, we propose a novel text encoder pre-training method **TEAMS** which stands for "Training ELECTRA Augmented with Multi-word Selection". Compared with ELECTRA, our method also consists of a generator and a discriminator but they are equipped with different pre-training tasks. For each masked position in the input sentence, the generator replaces the original token with an alternative token and samples a candidate set that consists of the original token and other K non-original ones. Then, we train the discriminator to simultaneously perform two tasks: (1) a *multi-word selection* task in which the discriminator learns to select the original token from the sampled candidate set, and (2) a *replaced token detection* task similarly defined in ELECTRA. The first task, as a $(K+1)-$way classification on the masked positions, pushes the discriminator to differentiate ground truth tokens from other negative non-original ones. At the same time, the second task, with reduced task complex-

ity, keeps the discriminator to achieve the same sample efficiency as ELECTRA.

To further improve the performance and efficiency of our method, we introduce two techniques. The first one is using attention-based task-specific heads for discriminator multi-task pre-training. Different from previous studies (Liu et al., 2019a; Sun et al., 2020) that pass the last encoder layer outputs to different task heads, our method directly incorporates task-specific attention layers into the discriminator encoder. Such a design offers higher flexibility in capturing task-specific token dependencies in sequence and leads to significant performance boost. The second technique is to share the bottom layers of the generator and the discriminator. This technique reduces the number of parameters, saves computes, and serves as a form of regularization that stabilizes the training and helps the generalization.

Combining above novelties all together, we train our models of various sizes and test their performance on the GLUE natural language understanding benchmark (Wang et al., 2019b) and SQuAD question answering benchmark (Rajpurkar et al., 2016). We show that TEAMS substantially outperforms previous MLM-based methods and ELECTRA, given the same model size and pre-training data. For example, our base-sized model, achieving 84.51 SQuAD 2.0 F1 score, outperforms BERT and ELECTRA by 8.34 and 2.99, respectively. Moreover, TEAMS-Base can outperform ELECTRA-Base++ using a fraction of computes.

Contributions. The major contributions of this paper are summarized as follows: (1) We propose a new text encoder pre-training method TEAMS that simultaneously learns a generator and a discriminator using multi-task learning. (2) We develop two techniques, attention-based task-specific head and partial layer sharing, to further improve TEAMS performance. (3) We conduct extensive experiments to verify the effectiveness of TEAMS on GLUE and SQuAD benchmarks[1].

2 Background

In this section, we first discuss some related studies on pre-training text encoders. Then, we introduce our notations and describe ELECTRA in details.

2.1 Text Encoder Pre-training

Current state-of-the-art natural language processing systems often rely on a text encoder to generate contextualized representations. This text encoder is commonly pre-trained on massive unlabeled corpora using different self-supervised tasks. Peters et al. (2018) and Radford et al. (2018) pre-train either a LSTM or a Transformer (Vaswani et al., 2017) using the standard language modeling task. To further improve pre-trained models, more effective pre-training objectives have been developed, including masked language modeling and next sentence prediction in BERT (Devlin et al., 2019), permutation language modeling in XLNet (Yang et al., 2019), masked span prediction in SpanBERT (Joshi et al., 2019), sentence order prediction in Struct-BERT (Wang et al., 2020), and more.

Most pre-training methods demand massive amounts of computes, which limits their accessibilities and raises concerns about their environmental costs. To alleviate such issue, Gong et al. (2019) and Yang et al. (2020) propose to accelerate BERT training by progressively stacking a shallow model to a deep model. Gu et al. (2020) extend this idea by growing a low-cost model in different dimensions. Along another line of work, Clark et al. (2020a) propose a new pre-training task, named replaced token detection, that learns a text encoder to distinguish real input tokens from synthetically generated replacements. Compared to BERT-style MLM pre-training in which only 15% of tokens are utilized, ELECTRA can leverage *all tokens* in input sentences and thus achieves better sample efficiency. Following this idea, Xu et al. (2020) propose a new pre-training task based on the multi-choice cloze test with a rejection option, and Clark et al. (2020b) connect ELECTRA with cloze modeling and pre-train the text encoder as an energy-based cloze model. As our method is built upon ELECTRA, we discuss it in more detail below.

2.2 ELECTRA

ELECTRA jointly trains two models, a generator G and a discriminator D. Both models adopt the Transformer architecture as their backbones and map a sentence of n tokens $\mathbf{x} = [x_1, \ldots, x_n]$ to their corresponding contextualized representations $h(\mathbf{x}) = [h(\mathbf{x})_1, \ldots, h(\mathbf{x})_n]$.

The generator G is trained using the masked language modeling (MLM) task. Specifically, given an input sequence $\mathbf{x}$, it first randomly selects a few

masked positions and replaces tokens at these positions with a special mask symbol [MASK]. We denote this masked sequence as $\mathbf{x}^M$. Then, the generator learns to predict the original identities of those masked-out tokens by minimizing the below MLM loss:

$$\mathcal{L}_{\text{MLM}}(\mathbf{x}; G) = \mathbb{E}\left(\sum_{i:x_i^M = \texttt{[MASK]}} -\log P_G(x_i|\mathbf{x}^M) \right), \tag{1}$$

where $P_G(x_i|\mathbf{x}^M)$ is the probability that G predicts token x_i appears in the masked position i in $\mathbf{x}^M$, and the expectation is taken over the random draw of masked positions. More specifically, the generator calculates $P_G(x_i|\mathbf{x}^M)$ by passing contextualized representations $h_G(\mathbf{x}^M)$ into a softmax layer as follows:

$$P_G(x_i|\mathbf{x}^M) = \frac{\exp(e(x_i)^T h_G(\mathbf{x}^M)_i)}{\sum_{x_i' \in V} \exp(e(x_i')^T h_G(\mathbf{x}^M)_i)}, \tag{2}$$

where $e(x_i)$ is the embedding of token x_i and V denotes the vocabulary of all tokens. Finally, for each masked position i, the generator samples one token $\hat{x}_i \sim P_G(\cdot|\mathbf{x}^M)$ and replaces the original token x_i with $\hat{x}_i$. We use $\mathbf{x}^R$ to denote this corrupted sentence with replaced tokens.

The discriminator D learns to perform the replaced token detection (RTD) task that requires a model to predict whether each token in $\mathbf{x}^R$ is replaced or not. In particular, ELECTRA adopts a sigmoid layer, on top of the discriminator output contextualized representations $h_D(\mathbf{x}^R)$, to decide the probability that token x_i^R matches the original token x_i as follows:

$$P_D(x_i^R = x_i) = \text{sigmoid}(w^T h_D(\mathbf{x}^R)_i), \tag{3}$$

where w is a learnable parameter. The loss on D is then defined as follows:

$$\mathcal{L}_{\text{RTD}}(\mathbf{x}, \mathbf{x}^R; D) = \mathbb{E}\Big(\sum_{i:x_i^R = x_i} -\log P_D(x_i^R = x_i)$$
$$+ \sum_{i:x_i^R \neq x_i} -\log(1 - P_D(x_i^R = x_i))\Big). \tag{4}$$

Finally, the generator and discriminator are jointly learned based on losses in Eq. (1) and Eq. (4). After pre-training, ELECTRA throws out the generator and keeps only the discriminator for fine-tuning on downstream tasks.

3 The TEAMS Method

In this section, we first introduce a new pre-training task named "multi-word selection". Then, we present our TEAMS method with two techniques for performance improvements.

3.1 Multi-word Selection Task

To train a model on an input sequence $\mathbf{x} = [x_1, \ldots, x_n]$ using the multi-world selection task, we first choose a random set of positions in this sequence, denoted as $\{i_1, \ldots, i_m\}$ where m is an integer between 1 and n. Then, for each chosen position $i_j, j \in \{1, \ldots, m\}$, we replace token x_{i_j} with another token $\hat{x}_{i_j}$ and create a *candidate set* S_{i_j} that includes the original token x_{i_j} and K non-original ones. Following ELECTRA, we use $\mathbf{x}^R$ to denote the corrupted sentence with all tokens in chosen positions replaced. Finally, the model inputs the corrupted sentence and outputs a probability for selecting the original token x_{i_j} from the candidate set S_{i_j} as follows:

$$P(x_{i_j}|\mathbf{x}^R, S_{i_j}) = \frac{\exp(e(x_{i_j})^T h(\mathbf{x}^R)_{i_j})}{\sum_{x_{i_j}' \in S_{i_j}} \exp(e(x_{i_j}')^T h(\mathbf{x}^R)_{i_j})}, \tag{5}$$

where $h(\mathbf{x}^R)_{i_j}$ is the contextualized representation of token $\hat{x}_{i_j}$ from the model outputs.

Figure 1 shows a concrete example wherein a sequence of 6 tokens is given and its 2nd, 4th, and 6th positions are chosen to be masked. Take the 2nd position as an example, the generator replaces the original token x_{i_1}="*famous*" with another token $\hat{x}_{i_1}$="*old*" and generates the candidate set S_{i_1}={"*top*", "*young*", "*french*", "*famous*"} which includes the original token x_{i_1} and $K = 3$ non-original alternatives.

We may view the multi-word selection task as a simplification of masked language modeling and a generalization of replaced token detection. Asking the model to select the correct word from a candidate set rather than from the entire vocabulary, we can save more computes. At the same time, being essentially a $(K + 1)-$way classification problem, the multi-word selection task is more challenging than the replaced token detection task (which is a binary classification problem) and thus pushes the model to learn more semantic representations. We describe how to generate the candidate set and present our entire method below.

3.2 Multi-task Learning in TEAMS

In TEAMS, we jointly train two transformer encoders, one as the generator network G and the other as the discriminator network D. Given a masked sequence $\mathbf{x}^M$, we use the generator G to perform two tasks for each masked position i_j in this sequence. First, similar to ELECTRA, we sample one token $\hat{x}_{i_j} \sim P_G(\cdot|\mathbf{x}^M)$ (c.f. Eq. (2)) and
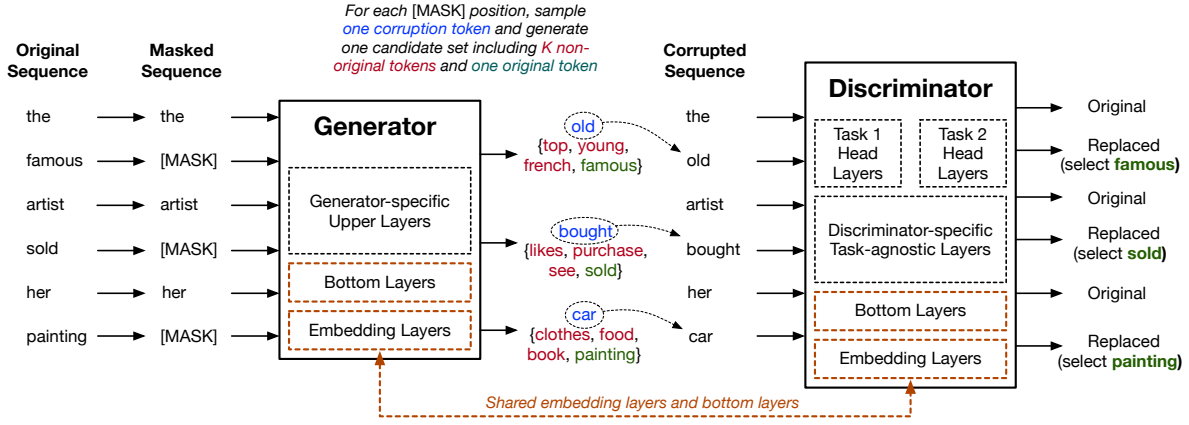
Figure 1: The overview framework of TEAMS. For each masked position, the generator replaces its original token with a new one and outputs a candidate set consisting of the original token and another K possible alternatives. The discriminator inputs the corrupted sentence and learns to (1) predict for every token whether it is replaced or not and (2) select the original token from the candidate set for each masked position.

obtain the corrupted sequence $\mathbf{x}^R$. Second, we draw K non-original tokens $\{x_{i_j}^1, \ldots, x_{i_j}^K\}$ from $P_G(\cdot|\mathbf{x}^M)$ without replacement[2] and construct the candidate set $S_{i_j} = \{x_{i_j}, x_{i_j}^1, \ldots, x_{i_j}^K\}$. Finally, we learn the generator G using the standard masked language modeling task (c.f. Eq. (1)).

On the discriminator side, we train the discriminator network D using two tasks — replaced token detection (RTD) task and multi-word selection (MWS) task. Given a corrupted sentence $\mathbf{x}^R$ of length n, the discriminator will generate two sets of contextualized representations $\{h_D^{\mathrm{RTD}}(\mathbf{x}^R)_i\}|_{i=1}^n$ and $\{h_D^{\mathrm{MWS}}(\mathbf{x}^R)_i\}|_{i=1}^n$, one for each pre-training task. For each position $i \in \{1, \ldots, n\}$, we use $h_D^{\mathrm{RTD}}(\mathbf{x}^R)_i$ to calculate the probability that the token $\mathbf{x}_i^R$ is replaced as follows:

$$P_D(\mathbf{x}_i^R = \mathbf{x}_i) = \mathrm{sigmoid}(w^T h_D^{\mathrm{RTD}}(\mathbf{x}^R)_i), \quad (6)$$

and optimize the same RTD loss defined in Eq. (4). For each masked position $i_j, j \in \{1, \ldots, m\}$, we obtain the candidate set S_{i_j} from generator outputs and use $h_D^{\mathrm{MWS}}(\mathbf{x}^R)_{i_j}$ to compute the probability of selecting the correct original token x_{i_j} from this candidate set as follows:

$$P_D(x_{i_j}|\mathbf{x}^R, S_{i_j}) = \frac{\exp(e(x_{i_j})^T h_D^{\mathrm{MWS}}(\mathbf{x}^R)_{i_j})}{\sum_{x_{i_j}' \in S_{i_j}} \exp(e(x_{i_j}')^T h_D^{\mathrm{MWS}}(\mathbf{x}^R)_{i_j})}. \quad (7)$$

As the multi-word selection task is a multi-class classification problem, we define its loss function

as follows:

$$\mathcal{L}_{\mathrm{MWS}}(\mathbf{x}, \mathbf{x}^R; D, \mathbb{S}) = \mathbb{E}\left(\sum_{j=1}^m - \log P_D(x_{i_j}|\mathbf{x}^R, S_{i_j})\right), \quad (8)$$

where $\mathbb{S} = \{S_{i_j}\}|_{j=1}^m$ is the collection of candidate sets at all masked positions. Finally, we learn TEAMS by optimizing a combined loss as follows:

$$\min_{G,D}\left(\mathcal{L}_{\mathrm{MLM}}(\mathbf{x}; G) + \lambda_1 \mathcal{L}_{\mathrm{RTD}}(\mathbf{x}, \mathbf{x}^R; D) \right.$$
$$\left. + \lambda_2 \mathcal{L}_{\mathrm{MWS}}(\mathbf{x}, \mathbf{x}^R; D, \mathbb{S})\right), \quad (9)$$

where λ_1 and λ_2 are two loss balancing hyper-parameters. For the example sequence in Figure 1, the discriminator needs to predict the tokens in 1st, 3rd, 5th positions are not replaced, the tokens in 2nd, 4th, 6th positions are replaced, and select tokens "*famous*", "*sold*", and "*painting*" in 2nd, 4th, 6th positions, respectively.

After pre-training, we keep the discriminator network and fine-tune it for downstream applications[3].

Attention-based Task-specific Heads. One remaining question is how to generate two sets of *task-specific* representations on the discriminator side. Previous studies (Liu et al., 2019a; Sun et al., 2020; Aroca-Ouellette and Rudzicz, 2020) achieve this goal by adding task-specific layers on top of each individual token, as shown in Figure 2 (Left). However, this approach does not model token dependencies within the task-specific layers.

In this work, we propose to use attention-based task-specific heads to capture global dependen-

---

[2]More discussions on other possible negative sampling strategies are presented in experiment section.

[3]Empirically, we find that using the contextualized representations for the MWS task (i.e., $\{h_D^{\mathrm{MWS}}(\mathbf{x}^R)_i\}|_{i=1}^n$) can achieve better fine-tuning performance.
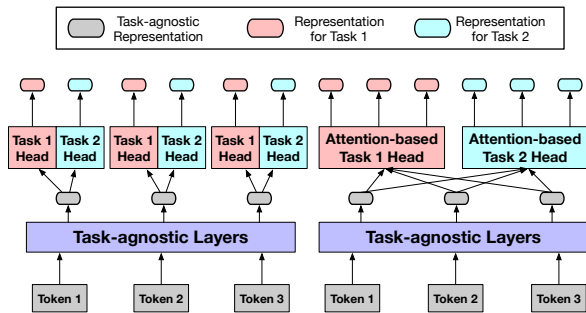
Figure 2: Architectures for transforming task-agnostic representations to task-specific representations. (Left) Adding task-specific heads on each token separately. (Right) Using task-specific attention heads capture all token information holistically.

cies in sequences. Particularly, we design this attention head to be one transformer layer (i.e., a self-attention block followed by a fully connected network with one hidden layer). Since our discriminator also uses a transformer model to obtain each token's task-agnostic representation, we can merge one task head into the discriminator backbone. From this perspective, we can generate different sets of *task-specific* representations as follows. First, we input the sequence to a transformer with L layers and retrieve the final layer output representations for one task. Then, we feed the output of an intermediate layer (*e.g.*, the $(L-1)^{th}$ layer) into another transformer layer to obtain token representations for the second task.

Partial Layer Sharing. ELECTRA has shown that tying the embedding layers of the generator and the discriminator can help improve the pre-training effectiveness. Our study confirms this observation and finds that sharing some transformer layers of the generator and discriminator and can further boost the model performance. More specifically, we design the generator to have the same "width" (i.e., hidden size, intermediate size and number of heads) as the discriminator and share the bottom half of all transformer layers between the generator and the discriminator.

4 Experiments

4.1 Experiment Setups

Pre-training Datasets. We use two datasets for model pre-training: (1) **WikiBooks**, which consists 3.3 Billion tokens from English Wikipedia and BooksCorpus (Zhu et al., 2015). This is the same dataset used in BERT (Devlin et al., 2019). (2) **WikiBooks++**, which extends WikiBooks dataset

to 33 Billion tokens by including data from Gigaword (Parker et al., 2011), ClueWeb (Callan et al., 2009), and CommonCrawl (Crawl, 2019). The same dataset is used in XLNet (Yang et al., 2019) and ELECTRA (Clark et al., 2020a).

Evaluation Datasets and Metrics. We evaluate all pre-trained models on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019b) and Stanford Question Answering (SQuAD) dataset (Rajpurkar et al., 2016). GLUE benchmark includes various tasks formatted as either single sentence classification (SST, CoLA) or sentence pair classification (*e.g.*, RTE, MNLI, QNLI, MRPC, QQP, STS). More details of each task are available in the Appendix Section A. SQuAD dataset requires models to select a text span from a given passage that answers a question. In SQuAD v1.1, the answers can always be located in the passage, while SQuAD v2.0 contains some questions unanswerable by the given passage.

We compute Spearman correlation for STS, Matthews correlation for CoLA, accuracy for all other GLUE tasks, and report the GLUE score as the average of all 8 tasks. For SQuAD, we use the standard evaluation metrics of Exact Match (EM) and F1 scores. Since different random seeds may significantly affect fine-tuned model performances, we report the median of 15 fine-tuning runs from the same pre-trained model checkpoint for each result. Unless stated otherwise, results are on the GLUE and SQuAD development sets.

Model Hyper-parameters. We follow and evaluate TEAMS with different model sizes. For *small-sized* model, we set model hidden dimension to 256 and reduce token embedding dimension to 128. The transformer in the discriminator network has 12 layers and each layer consists of 4 attention heads with the intermediate layer size 1024. For *base-sized* model, we adopt the commonly used BERT-base configuration with 768 hidden dimension, 12 layers with 12 attention heads, and 3072 intermediate layer size. For *large-sized* model, we use BERT-large configuration with 1024 hidden dimension, 24 layers with 16 attention heads, and 4096 intermediate layer size. Following (Clark et al., 2020a), we design the generator network size to be 1/2 of the discriminator network size for models of all sizes. For TEAMS, we set the loss balancing parameter $\lambda_1 = 5$, $\lambda_2 = 2$ (c.f., Eq. (9)), and the number of sampled non-original tokens $K = 5$ (c.f., Section 3.1).

Method	Params	GLUE	SQuAD 1.1		SQuAD 2.0	
			EM	F1	EM	F1
BERT-Small	14M	78.52	76.30	84.39	68.95	71.79
ELECTRA-Small (Our reimplementation)	14M	80.36	76.50	84.67	69.17	71.68
TEAMS-Small	14M	**80.70**	**78.84**	**86.40**	**72.33**	**75.24**
BERT-Small++	14M	79.10	76.48	84.75	68.37	71.01
ELECTRA-Small++ (Our reimplementation)	14M	81.71	77.45	85.32	70.07	72.91
ELECTRA-Small++ (Public checkpoint re-evaluate)	14M	81.24	77.62	85.63	71.12	73.95
TEAMS-Small++	14M	**81.99**	**78.94**	**86.65**	**72.11**	**75.11**
BERT-Base	110M	83.46	80.62	88.16	73.26	76.17
ELECTRA-Base (Our reimplementation)	110M	84.63	83.87	90.64	78.59	81.52
TEAMS-Base	110M	**85.57**	**85.21**	**91.69**	**81.59**	**84.51**
BERT-Base++	110M	84.26	84.48	91.08	78.83	81.72
ELECTRA-Base++ (Our reimplementation)	110M	86.29	85.09	91.65	81.31	84.04
ELECTRA-Base++ (Public checkpoint re-evaluate)	110M	87.13	85.09	91.68	79.16	82.06
TEAMS-Base++	110M	**87.16**	**86.05**	**92.48**	**82.73**	**85.59**
BERT-Large	335M	84.91	86.35	92.61	82.19	84.78
ELECTRA-Large (Our reimplementation)	335M	89.20	88.79	94.50	86.02	88.72
ELECTRA-Large (Public checkpoint re-evaluate)	335M	89.38	88.76	94.49	86.79	89.56
TEAMS-Large	335M	**89.44**	**88.86**	**94.61**	**87.08**	**89.86**

Table 1: Comparison results of TEAMS and baseline methods on GLUE and SQuAD datasets. All results are the medians of 15 fine-tuning runs with different initial random seeds. As ELECTRA original paper only releases the public checkpoints for Small++, Base++, and Large models, we can only report results for these three variants.

During pre-training, we set the batch size to be 256 and the input sequence length to be 512 for both small-sized and base-sized models. We update small-sized models for 500K steps and base-sized models for 1M steps on the WikiBooks dataset. Moreover, we test the performance of each model when it is pre-trained for longer time with larger batch size using the WikiBooks++ dataset. We use the suffix "*small++*" to denote a small-sized model pre-trained for 2M steps with batch size 256, and the suffix "*base++*" to denote a base-sized model pre-trained for 1M steps with batch size 1024. Finally, for large-sized models, we use batch size 2048 and pre-train the model for 1.76M steps. All large-sized models and models with suffix "++" are trained using the WikiBooks++ dataset. More pre-training and fine-tuning details are included in the Appendix Section B and C.

Model Implementations. For fair comparison, we implement all compared methods in TensorFlow 2 and evaluate their performances using the official pipeline in TensorFlow Model Garden[4]. In addition to our own implementations, we also report the performance of ELECTRA publicly released checkpoints[5]. All models are trained on TPU v3.

---

[4]https://github.com/tensorflow/models.
[5]https://github.com/google-research/electra.

4.2 Experiment Results

We validate the advantages of our proposed TEAMS method by comparing it with BERT (Devlin et al., 2019) and ELECTRA (Clark et al., 2020a). Table 1 shows the comparison results on GLUE and SQuAD datasets. We find that TEAMS can consistently outperform baseline models of the same size. For example, compared to ELECTRA-Base, our TEAMS-Base improves SQuAD 2.0 performance from 78.59 to 81.59 and from 81.52 to 84.51 in terms of EM and F1 score, respectively.

To further verify the performance improvements do not come from consuming more computations, we draw the learning curves of TEAMS-Small/Base and ELECTRA-Small/Base in Figure 3. We observe that for both small-sized and base-sized models, our method can consistently outperform ELECTRA when trained for the some amount of time. Moreover, on SQuAD datasets, TEAMS-Base can even outperform the ELECTRA-Base++ model that requires much more computation.

4.3 Ablation Studies

We continue to evaluate the design of each component within TEAMS and test its sensitivity to some critical hyper-parameters.

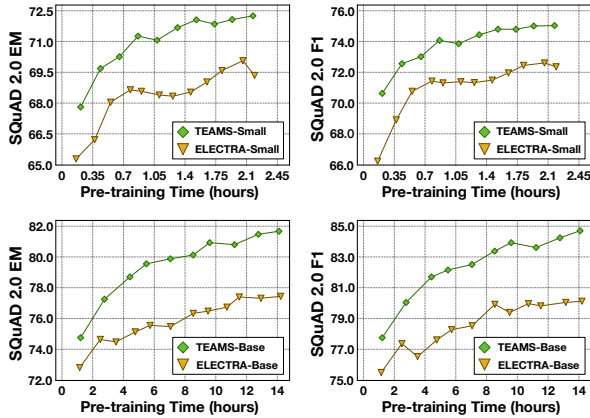**Effectiveness of Pre-training Tasks.** We report the results of small-sized models learned using dif-

Figure 3: Learning curves for small-sized and base-sized models.

Pre-training Task(s)	GLUE	SQuAD 1.1		SQuAD 2.0	
		EM	F1	EM	F1
Only MLM (i.e., BERT)	79.10	76.48	84.75	68.37	71.01
Only RTD (i.e., ELECTRA)	81.71	77.45	85.32	70.07	72.91
Only MWS	79.65	77.30	85.32	70.10	72.80
RTD + MWS (i.e., TEAMS)	**81.99**	**78.94**	**86.65**	**72.11**	**75.11**

Table 2: Effectiveness of multi-task pre-training for *small++* models. "MLM", "RTD", and "MWS" stand for "masked language modeling", "replaced token detection", and "multi-word selection", respectively.

ferent pre-training tasks in Table 2. First, we can see that the model trained with multi-word selection (MWS) task can outperform the one learned using masked language modeling (MLM) task. Second, on SQuAD datasets, we find that pre-training on only 15% of masked tokens using MWS task is comparable with pre-training on all tokens using replaced token detection (RTD) task. These observations demonstrate the effectiveness of our proposed MWS task. Finally, we show that a text encoder pre-trained using both MWS and RTD tasks can outperform those learned using only single task.

Task-specific Layer Designs. In TEAMS, we pre-train the discriminator network using multi-task learning and introduce the attention-based task-specific heads. To verify the effectiveness of these attention-based task-specific heads, we train another model that uses the traditional feed forward network (FFN) as the task-specific head. Table 3 shows the results. We can see that our model achieves better performances because the attention-based heads can effectively model the token dependencies in sequences.

We continue to study where to add these task-specific heads. Currently, given a transformer with 12 layers, we treat its last layer output for one task

Method	GLUE	SQuAD 1.1		SQuAD 2.0	
		EM	F1	EM	F1
ELECTRA-Small++	81.71	77.45	85.32	70.07	72.91
TEAMS-Small++	81.99	78.94	86.65	72.11	75.11
Use FFN task heads	81.49	78.18	86.35	72.00	74.90
Add task head on 12th layer	81.29	79.08	86.66	72.47	75.31
Use RTD task head outputs	81.83	77.72	85.80	69.56	72.57

Table 3: Analysis of task-specific layers and exported representations for *small++* models. Please refer to Section 4.3 for detailed descriptions of each method.

and feed the 11th layer output to a separate transformer layer to obtain representations for the second task[6]. An alternative design is to add two separate transformer layers (as two task-specific heads) directly on top of the last layer (i.e., the 12th layer). As shown in Table 3, we find the latter design can slightly improve the model performance on SQuAD datasets but leads to a larger discriminator network with effectively 13 transformer layers and thus requires more computation during both pre-training and fine-tuning stages.

Finally, as our discriminator network will output two sets of contextualized representations, one for MWS task and the other for RTD task, we need to decide which set of representations to use in the fine-tuning stage. Empirically, we find the representations for MWS task has better fine-tuning performance than the ones for RTD task, especially on the SQuAD datasets (c.f. Table 3). This observation also confirms the effectiveness of our proposed MWS task as it produces representations capturing more fine-grained semantic information compared to the RTD task.

Partial Layer Sharing. Table 4 reports the results of our models with different levels of parameter sharing between the generator and the discriminator. First, we can see that tying all generator layers with discriminator layers results in significant performance drops, as such a binding restricts the model representation power. Second, we find that compared to no weight sharing, our design of partial layer tying can improve the model performance. One possible explanation is that such layer tying serves as an implicit form of regularization and forces the shared transformer layers to capture useful information for both generator pre-training task (i.e., MLM) and discriminator pre-training tasks (i.e., RTD and MWS).

---

[6]We can interpret the first 11 layers as *task-agnostic* layers and view the 12th layer and the newly introduced separate layers as two *task-specific* heads.

Method	GLUE	SQuAD 1.1 EM	SQuAD 1.1 F1	SQuAD 2.0 EM	SQuAD 2.0 F1
ELECTRA-Small++	81.71	77.45	85.32	70.07	72.91
TEAMS-Small++	**81.99**	**78.94**	**86.65**	72.11	75.11
Full Tie	80.57	77.75	85.76	69.93	72.82
No Tie	81.65	78.42	86.32	**72.73**	**75.73**
ELECTRA-Base++	86.29	85.09	91.65	81.31	84.04
TEAMS-Base++	**87.16**	**86.05**	**92.48**	**82.73**	**85.59**
No Tie	86.63	85.51	91.98	80.72	83.60

Table 4: Effect of sharing generator and discriminator bottom layers for *small++* and *base++* models. "Full Tie" and "No Tie" stand for tying all or none of generator layers with the discriminator, respectively.

Sampling Strategy and Negative Sample Size. To use the multi-world selection task for pre-training, we need to first obtain a set of negative samples (i.e., non-original tokens) for each masked position in a sequence. In this study, we test two strategies to generate K negative samples for each masked position. Given the generator output probability distribution for a target position, we can either sample from this distribution K times without replacement or directly select K non-original tokens with the highest probabilities. We denote these two approaches as "Sampled" and "Hardest", respectively, and report the results in Figure 5. First, we can see that performing repeated sampling is a better strategy than always selecting those hardest samples. One possible reason is that the "Sampled" strategy can generate more diverse negative samples and thus helps model to generalize[7]. Second, we notice that increasing K over 5 will somewhat hurt the model performance. One reason is that a larger K causes a higher probability of including false negative examples. Finally, we find that for a wide range of K from 3 to 50, our method can outperform ELECTRA, which further demonstrates the effectiveness of multi-word selection task.

Generator Size. We test how the size of generator affects the model performance by varying the number of transformer layers in the generator. For all tested models, we tie the bottom half of generator with the discriminator. Figure 4 reports the results. We find that the performance first increases as the generator size increases until it reaches about half of the discriminator size and then starts to decrease when we further increase the generator size. The same results also hold for base-sized models.

---

[7]A similar result is also witnessed in (Shen et al., 2019) and thus we adopt the "Sampled" approach in this study.
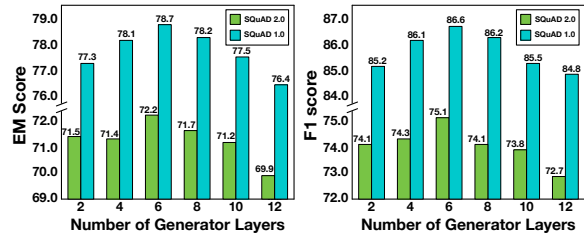

Figure 4: Analysis of the number of generator layers in TEAMS-small++ models.
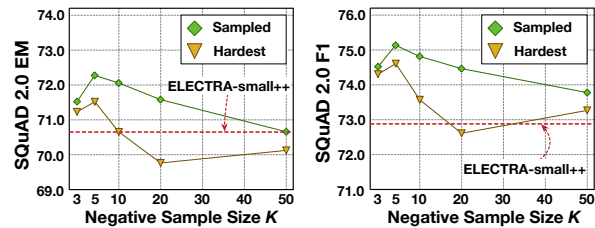

Figure 5: Analysis of negative sampling strategies and negative sample sizes for *small++* models. Based on the generator output distribution, we either perform weighted sampling without replacement K times ("Sampled") or select K most likely non-original tokens ("Hardest") for each masked position.

5 Related Work

Besides the general language pre-training work we discussed in Section 2.1, this study is particularly related to methods that apply multi-task learning (Caruana, 1997; Ruder, 2017; Shen et al., 2018) to language representation learning. An early study (Liu et al., 2019a) proposes to simultaneously fine-tune a pre-trained BERT model to perform multiple natural language understanding tasks and achieves promising results on the GLUE dataset. Sun et al. (2020) continue this line of work and propose to push the multi-task learning to the model pre-training stage. Specifically, they use a continual multi-task learning framework that incrementally builds and inserts seven auxiliary tasks (*e.g.*, masked entity prediction, sentence distance prediction, *etc.*) to the text encoder. More recently, Aroca-Ouellette and Rudzicz (2020) extend this idea to incorporate fourteen auxiliary tasks and identify six tasks are particularly useful. While achieving inspiring performance, these studies all assume the MLM pre-training task must present and just combine MLM with additional tasks. In this paper, we relax this assumption and combine our new multi-word selection task with the replace token detection task for effective pre-training.

6 Conclusions and Future Work

This work presents a new text encoder pre-training method that simultaneously learns a generator and a discriminator using multi-task learning. We propose a new pre-training task, multi-word selection, and combine it with previous pre-training tasks for efficient encoder pre-training. We also develop two techniques, attention-based task-specific heads and partial layer sharing, to further improve pre-training effectiveness. Extensive experiments on GLUE and SQuAD datasets demonstrate our TEAMS method can consistently outperform previous state-of-the-arts methods. In the future, we plan to explore how other auxiliary pre-training tasks can be integrated into our framework. Moreover, we consider extending our pre-training method to text encoders with other architectures such as those based on dynamic convolution and sparse attention. Finally, being orthogonal to this study, distillation techniques could be applied to further compress our pre-trained encoders into smaller models for faster inference speeds.

Broader Impact Statement

Recent years have witnessed the great success of pre-trained text encoders in lots of NLP applications such as text classification, question answering, text retrieval, dialogue system, etc. This paper presents a new pre-training method TEAMS that learns a text encoder with better performance using lower training cost. Therefore, on the positive side, our work has the potentials to benefit all downstream applications that leverage a pre-trained text encoder, especially those applications with limited computation resources. On the negative side, TEAMS, as one specific pre-training method, could still face the generic issues for all language pre-training work. For example, the pre-training large corpora, collected from the internet, may include abusive language usages and fail to capture the cultures that have smaller linguistic footprints online.

References

Stephane T Aroca-Ouellette and F. Rudzicz. 2020. On losses for modern language models. In *EMNLP*.

Jamie Callan, Mark Hoy, Changkuk Yoo, and Le Zhao. 2009. Clueweb09 data set.

Rich Caruana. 1997. Multitask learning. *Machine learning*, 28(1):41–75.

Daniel M. Cer, Mona T. Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *SemEval@ACL*.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020a. Electra: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020b. Pre-training transformers as energy-based cloze models. In *EMNLP*.

Common Crawl. 2019. Common crawl. *URl: http://http://commoncrawl. org*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *IWP@IJCNLP*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B. Dolan. 2007. The third pascal recognizing textual entailment challenge. In *ACL-PASCAL@ACL*.

Linyuan Gong, D. He, Zhuohan Li, T. Qin, Liwei Wang, and T. Liu. 2019. Efficient training of bert by progressively stacking. In *ICML*.

X. Gu, Liyuan Liu, H. Yu, Jing Li, Chen Chen, and J. Han. 2020. On the transformer growth for progressive bert training. *ArXiv*, abs/2010.12562.

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. 2017. First Quora dataset release: Question pairs.

Zihang Jiang, Weihao Yu, Daquan Zhou, Y. Chen, Jiashi Feng, and S. Yan. 2020. Convbert: Improving bert with span-based dynamic convolution. In *NeurIPS*.

Mandar Joshi, Danqi Chen, Y. Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2019. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

H. Levesque, E. Davis, and L. Morgenstern. 2011. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Xiaodong Liu, Pengcheng He, W. Chen, and Jianfeng Gao. 2019a. Multi-task deep neural networks for natural language understanding. In *ACL*.

Y. Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, M. Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Yang Liu and Mirella Lapata. 2019. Text summarization with pretrained encoders. In *EMNLP/IJCNLP*.

R Parker, D Graff, J Kong, K Chen, and K Maeda. 2011. English gigaword fifth edition ldc2011t07 (tech. rep.). Technical report, Technical Report. Linguistic Data Consortium, Philadelphia.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL-HLT*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

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

Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *ArXiv*, abs/1706.05098.

J. Shen, Maryam Karimzadehgan, Michael Bendersky, Zhen Qin, and Donald Metzler. 2018. Multi-task learning for email search ranking with auxiliary query clustering. In *CIKM*.

J. Shen, Ruiliang Lyu, Xiang Ren, M. Vanni, Brian M. Sadler, and Jiawei Han. 2019. Mining entity synonyms with efficient neural set generation. In *AAAI*.

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 *EMNLP*.

Y. Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, H. Wu, and Haifeng Wang. 2020. Ernie 2.0: A continual pre-training framework for language understanding. In *AAAI*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, L. Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. Superglue: A stickier benchmark for general-purpose language understanding systems. In *NeurIPS*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*.

Wei Wang, B. Bi, Ming Yan, Chen Wu, Zuyi Bao, Liwei Peng, and L. Si. 2020. Structbert: Incorporating language structures into pre-training for deep language understanding. In *ICLR*.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2018. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*.

Zhenhui Xu, Linyuan Gong, Guolin Ke, Di He, Shu xin Zheng, Liwei Wang, Jiang Bian, and T. Liu. 2020. Mc-bert: Efficient language pre-training via a meta controller. *ArXiv*, abs/2006.05744.

Cheng Yang, Shengnan Wang, Chao Yang, Yuechuan Li, Ru He, and Jingqiao Zhang. 2020. Progressively stacking 2.0: A multi-stage layerwise training method for bert training speedup. *ArXiv*, abs/2011.13635.

Z. Yang, Zihang Dai, Yiming Yang, J. Carbonell, R. Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*.

Y. Zhu, Ryan Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 19–27.

A GLUE Details

The original GLUE benchmark (Wang et al., 2019b) contains 9 natural language understanding datasets. We describe them below:

- **MNLI:** The Multi-genre Natural Language Inference Corpus (Williams et al., 2018) contains 393K training sentence pairs with textual entailment annotations. Given a premise sentence and a hypothesis sentence, a model needs to predict whether the premise entails the hypothesis, contradicts the hypothesis, or neither.

- **RTE:** Recognizing Textual Entailment (Giampiccolo et al., 2007) dataset is similar to MNLI and contains 2.5K sentence pairs with binary entailment annotations (entailment or contradiction).

- **QNLI:** Question Natural Language Inference dataset is a binary sentence pair classification dataset constructed from SQuAD (Rajpurkar et al., 2016). It contains 108K training sentence pairs and requires a model to predict whether a context sentence contains the answer to a question sentence.

- **CoLA:** Corpus of Linguistic Acceptability (Warstadt et al., 2018). This dataset includes 8.5K training sentences annotated with whether it is a grammatical English sentence.

- **SST:** Stanford Sentiment Treebank (Socher et al., 2013) dataset contains 67K sentences from movie reviews and their corresponding binary sentiment annotations.

- **MRPC:** Microsoft Research Paraphrase Corpus (Dolan and Brockett, 2005) includes 3.7K sentence pairs from online news sources. The task is to predict whether two sentences are semantically equivalent or not.

- **STS:** Semantic Textual Similarity (Cer et al., 2017) benchmark contains 5.8K training sentence pairs. The task is to predict the similarity score of two sentences from 1 to 5.

- **QQP:** Quora Question Pairs (Iyer et al., 2017) dataset includes 364K question pairs sampled from the community question-answering website Quora. Models are trained to predict whether a pair of questions are semantically equivalent.

- **WNLI:** Winograd NLI (Levesque et al., 2011) is a small natural language inference dataset. However, as GLUE official website[8] indicates there are some issues during its construction process, we follow previous studies (Clark et al., 2020a; Jiang et al., 2020) and exclude this dataset for fair comparisons.

B Pre-training Details

For the pre-training architecture configurations, we mostly use the same hyper-parameters as BERT and ELECTRA. To generate masked positions, we follow BERT and duplicate training data 40 times so each sequence is masked in 40 different ways. We find this static masking strategy performs similar to the dynamic masking strategy in ELECTRA, while being easier to implement and has less computation overhead. Besides, we notice a mask percentage of 15 works well for all models and thus do not increase it to 25 for large-size models as suggested in ELECTRA. We set λ_1 and λ_2, the loss balancing parameters to 5 and 2, respectively, to ensure different loss terms are of the same scale. For small-size and base-size models, we search for the learning rate out of {1e-4, 2e-4, 3e-4, 5e-4}, batch size from {128, 256, 512, 1024}, and training steps from {500K, 1M, 1.5M, 2M}. For large-size models, we search for the learning rate out of {1e-4, 2e-4, 3e-4, 5e-4} and batch size from {1024, 2048}. Also, we select the generator size out of {1/4, 1/3, 1/2} in early experiments. The best configurations are reported in the main text and we perform no other hyper-parameter tuning. The full set of hyper-parameters are listed in Table 5.

C Fine-tuning Details

For fair comparisons, we fine-tune all pre-trained checkpoints using the official pipeline in TensorFlow Model Garden[9] and report the median of 15 fine-tuning runs. We do not include layer-wise learning-rate decay. We search for the learning rate from {1e-5, 3e-5, 5e-5, 8e-5, 1e-4}, batch size from {32, 48}, and training epoch from {2, 3, 5}. For GLUE tasks, best evaluation scores during fine-tuning are reported. For SQuAD, scores at the end of fine-tuning are reported. The full set of hyper-parameters are listed in Table 6.

---

[8] https://gluebenchmark.com/faq
[9] https://github.com/tensorflow/models

Hyper-parameter	Small/Small++	Base/Base++	Large
Number of Layers	12	12	24
Embedding Dim.	128	768	1024
Hidden Dim.	256	768	1024
Intermediate Layer Dim.	1024	3072	4096
Number of Attention Heads	4	12	16
Attention Head Dim.	64	64	64
Generator Size (Multiplier for Number of Layers)	1/2	1/2	1/2
Mask Percentage	15	15	15
Learning Rate Decay	Linear	Linear	Linear
Warmup Steps	10000	10000	10000
Learning Rate	5e-4	2e-4/3e-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	256/256	256/1024	2048
Train Steps	500K/2M	1M/1M	1.76M

Table 5: Pre-training hyper-parameters.

Hyper-parameter	Value
Learning Rate	1e-4 in Small/Small++, 3e-5 in Base/Base++/Large for GLUE
	8e-5 in Small/Small++, 5e-5 in Base/Base++, and 3e-5 in Large for SQuAD 1.1/2.0
Learning Rate decay	Linear
Warmup fraction	0.1
Adam ϵ	1e-6
Adam β_1	0.9
Adam β_2	0.999
Dropout	0.1
Batch Size	32 for GLUE, 48 in Small/Small++/Large and 32 in Base/Base++ for SQuAD 1.1/2.0
Training Epochs	5 for GLUE, 5 in Small/Small+ and 2 in Base/Base++/Large for SQuAD 1.1/2.0

Table 6: Fine-tuning hyper-parameters.