ELECTRA: PRE-TRAINING TEXT ENCODERS AS DISCRIMINATORS RATHER THAN GENERATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

While masked language modeling (MLM) pre-training methods such as BERT produce excellent results on downstream NLP tasks, they require large amounts of compute to be effective. These approaches corrupt the input by replacing some tokens with `[MASK]` and then train a model to reconstruct the original tokens. As an alternative, we propose a more sample-efficient pre-training task called replaced token detection. Instead of masking the input, our approach corrupts it by replacing some input tokens with plausible alternatives sampled from a small generator network. Then, instead of training a model that predicts the original identities of the corrupted tokens, we train a discriminative model that predicts whether each token in the corrupted input was replaced by a generator sample or not. Thorough experiments demonstrate this new pre-training task is more efficient than MLM because the model learns from *all* input tokens rather than just the small subset that was masked out. As a result, the contextual representations learned by our approach substantially outperform the ones learned by methods such as BERT and XLNet given the same model size, data, and compute. The gains are particularly strong for small models; for example, we train a model on one GPU for 4 days that outperforms GPT (trained using 30x more compute) on the GLUE natural language understanding benchmark. Our approach also works well at scale, where we match the performance of RoBERTa, the current state-of-the-art pre-trained transformer, while using less than 1/4 of the compute.

MLM 用 [MASK] 代替输入中的某些token，并通过模型重建原始的toekn。

本文提出了新的预训练任务：replaced token detector。

代替mask掉一些词，这种做法是用从一个小的生成器网络中采样的词来代替输入中的某些词，，然后再用一个判别模型预测输入中的每个词是否是被替换的。

新的预训练任务相比MLM的优势在于，它可以学习所有的输入token，而不是仅被mask掉的一小部分词。

在相同的模型规模、数据、算力的条件下，该方法学到的上下文表示比BERT和XLNET更好。

1 INTRODUCTION

Current state-of-the-art representation learning methods for language can be viewed as learning denoising autoencoders (Vincent et al., 2008). They select a small subset of the unlabeled input sequence (typically 15%), mask the identities of those tokens (see BERT; Devlin et al. (2019)) or attention to those tokens (see XLNet; Yang et al. (2019)), and then train the network to recover the original input. While more effective than language-model pre-training due to learning bidirectional representations, these masked language modelin (MLM) approaches incur a substantial compute cost because the network only learns from 15% of the tokens per example.

As an alternative, we propose *replaced token detection*, a pre-training task in which the model learns to distinguish input tokens from plausible alternatives. Instead of masking, our method corrupts the input by replacing some tokens with samples from a proposal distribution, which is typically the output of a small masked language model. This corruption procedure solves a mismatch in BERT (although not in XLNet) where the network sees artificial `[MASK]` tokens during pre-training but not when being fine-tuned on downstream tasks. We then pre-train the network as a discriminator that predicts for every token whether it is an original or a replacement. In contrast, MLM trains the network as a generator that predicts the original identities of the corrupted tokens. One advantage of our discriminative task is that the model learns from *all* input tokens instead of just the small masked-out subset, making it more computationally efficient. Although our approach is reminiscent of training the discriminator of a GAN, our method is not "adversarial" in that the generator producing corrupted tokens is trained with maximum-likelihood due to the difficulty of applying GANs to text (Caccia et al., 2018).

replaced token detection 任务还能解决BERT中训练和微调不一致的问题。

We call our approach ELECTRA for "Efficiently Learning an Encoder that Classifies Token Replacements Accurately." As in prior work, we apply it to train Transformer text encoders (Vaswani

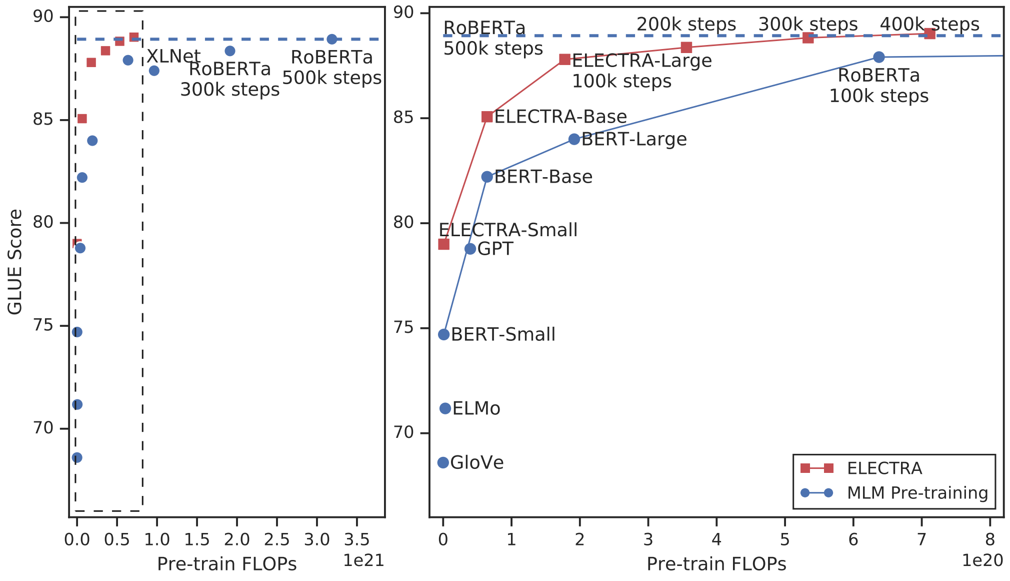

Figure 1: ELECTRA consistently outperforms MLM pre-training given the same compute budget. The right figure is a zoomed-in view of the dashed box.

et al., 2017) that can be fine-tuned on downstream tasks. Through a series of ablations, we show that learning from all input tokens causes ELECTRA to train much faster than BERT. We also show ELECTRA achieves higher accuracy on downstream tasks when fully trained.

Most current pre-training methods require large amounts of compute to be effective, raising concerns about their cost and accessibility. Since pre-training with more compute almost always results in better downstream accuracies, we argue an important consideration for pre-training methods should be compute efficiency as well as absolute downstream performance. From this viewpoint, we train ELECTRA models of various sizes and evaluate their performance on the GLUE natural language understanding benchmark (Wang et al., 2019) vs. their compute requirement. ELECTRA significantly outperforms MLM-based methods such as BERT and XLNet given the same model size, data, and compute (see Figure 1). For example, we build an ELECTRA-Small model that can be trained on 1 GPU in 4 days.[1] ELECTRA-Small outperforms a comparably small BERT model by 5 points on GLUE, and even outperforms the much larger GPT model (Radford et al., 2018). Our approach also works well at large scale, where we train a model that matches the performance of RoBERTa (Liu et al., 2019), the current state-of-the-art pre-trained Transformer, despite having fewer parameters and using 1/4 of the compute for training. Taken together, our results indicate that the discriminative task of distinguishing real data from challenging negative samples is more compute-efficient and parameter-efficient than existing generative approaches for language representation learning.

2 Method

We first describe the replaced token detection pre-training task; see Figure 2 for an overview. We suggest and evaluate several modeling improvements for this method in Section 3.2.

Our approach trains two neural networks, a generator G and a discriminator D. Each one primarily consists of an encoder (e.g., a Transformer network) that maps a sequence on input tokens $\boldsymbol{x} = [x_1, ..., x_n]$ into a sequence of contextualized vector representations $h(\boldsymbol{x}) = [h_1, ..., h_n]$. For a given position t, the generator outputs a probability for generating the token x_t with a softmax layer:

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

[1]It has 1/20th the parameters and requires 1/135th the pre-training compute of BERT-Large.

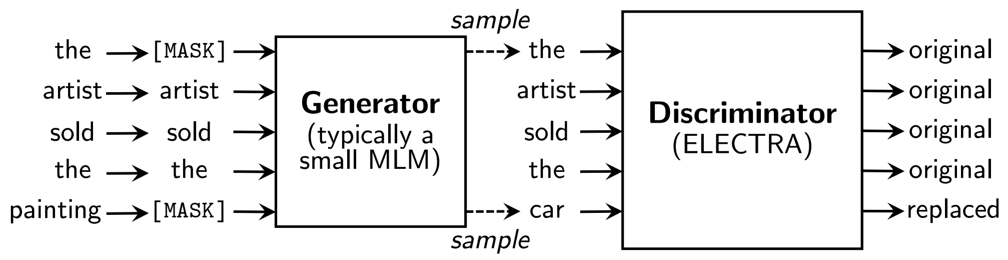

Figure 2: An overview of replaced token detection. The generator can be any model that produces an output distribution over tokens, but we usually use a small masked language model that is trained jointly with the discriminator. Although the models are structured like in a GAN, we train the generator with maximum likelihood rather than adversarially due to the difficulty of applying GANs to text. After pre-training, we throw out the generator and only fine-tune the discriminator on downstream tasks.

where e denotes token embeddings. For a given position t, the discriminator predicts whether the token x_t is "fake," i.e., that it comes from the generator rather than the data distribution:

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

The generator is trained to perform masked language modeling (MLM). Given an input $\boldsymbol{x} = [x_1, x_2, ..., x_n]$, MLM first select a random set of positions (integers between 1 and n) to mask out $\boldsymbol{m} = [m_1, ..., m_k]$.[2] The tokens in the selected positions are replaced with a `[MASK]` token: we denote this as $\boldsymbol{x}^{\text{masked}} = \text{REPLACE}(\boldsymbol{x}, \boldsymbol{m}, \texttt{[MASK]})$. The generator then learns to maximize the likelihood of the masked-out tokens. The discriminator is trained to distinguish tokens in the data from tokens that have been replaced by generator samples. More specifically, we create a corrupted example $\boldsymbol{x}^{\text{corrupt}}$ by replacing the masked-out tokens with generator samples and train the discriminator to predict which tokens in $\boldsymbol{x}^{\text{corrupt}}$ match the original input $\boldsymbol{x}$. Formally, model inputs are constructed according to

M向量存储被mask掉的位置。

$$m_i \sim \text{unif}\{1, n\} \text{ for } i = 1 \text{ to } k \qquad \hat{x}_i \sim p_G(x_i|\boldsymbol{x}^{\text{masked}}) \text{ for } i \in \boldsymbol{m}$$
$$\boldsymbol{x}^{\text{masked}} = \text{REPLACE}(\boldsymbol{x}, \boldsymbol{m}, \texttt{[MASK]}) \qquad \boldsymbol{x}^{\text{corrupt}} = \text{REPLACE}(\boldsymbol{x}, \boldsymbol{m}, \hat{\boldsymbol{x}})$$

and the loss functions are

$$\boxed{\begin{aligned}\mathcal{L}_{\text{MLM}}(\boldsymbol{x}, \theta_G) &= \mathbb{E}\left(\sum_{i \in \boldsymbol{m}} -\log p_G(x_i|\boldsymbol{x}^{\text{masked}})\right)\\ \mathcal{L}_{\text{Disc}}(\boldsymbol{x}, \theta_D) &= \mathbb{E}\left(\sum_{t=1}^{n} \mathbb{1}(x_t^{\text{corrupt}} = x_t)\log D(\boldsymbol{x}^{\text{corrupt}}, t) + \mathbb{1}(x_t^{\text{corrupt}} \neq x_t)\log(1 - D(\boldsymbol{x}^{\text{corrupt}}, t))\right)\end{aligned}}$$

Although similar to the training objective of a GAN, there are several key differences. First, if the generator happens to generate the correct token, that token is considered "real" instead of "fake"; we found this formulation to moderately improve results on downstream tasks. More importantly, the generator is trained with maximum likelihood rather than being trained adversarially to fool the discriminator. Adversarially training the generator is challenging because it is impossible to back-propagate through sampling from the generator. Although we experimented circumventing this issue by using reinforcement learning to train the generator (see Appendix D), this performed worse than maximum-likelihood training. Lastly, we do not supply the generator with a noise vector as input, as is typical with a GAN.

We minimize the combined loss

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

over a large corpus $\mathcal{X}$ of raw text. We approximate the expectations in the losses with a single sample. We don't back-propagate the discriminator loss through the generator (indeed, we can't because of the sampling step). After pre-training, we throw out the generator and fine-tune the discriminator on downstream tasks.

[2]Typically $k = \lceil 0.15n \rceil$, i.e., 15% of the tokens are masked out.

3 Experiments

3.1 Experimental Setup

评估集

We evaluate on the General Language Understanding Evaluation (GLUE) dataset (Wang et al., 2019). GLUE tasks cover textual entailment (RTE and MNLI) question-answer entailment (QNLI), paraphrase (MRPC), question paraphrase (QQP), textual similarity (STS), sentiment (SST), and linguistic acceptability (CoLA). Our evaluation metrics are Spearman correlation for STS, Matthews correlation for CoLA, and accuracy for the other tasks; we use "GLUE score" to refer to the average score over all tasks. We hope to evaluate on other datasets such as SQuAD (Rajpurkar et al., 2016) in the future. For most experiments we pre-train on the same data as BERT, which consists of 3.3 Billion tokens from Wikipedia and BooksCorpus (Zhu et al., 2015). However, for our Large model we pre-trained on the data used for XLNet (Yang et al., 2019), which extends the BERT dataset to 33B tokens by including data from ClueWeb (Callan et al., 2009), CommonCrawl and Gigaword (Parker et al., 2011). All of the pre-training and evaluation is on English data, although we think it would be interesting to apply our methods to multilingual data in the future. Our model architecture and most hyperparameters are the same as BERT's, see the Appendix for details. Some of our evaluation datasets are small, which means accuracies of fine-tuned models can vary substantially depending on the random seed. We therefore report the median of 10 fine-tuning runs from the same pre-trained checkpoint for each result. Unless stated otherwise, results are on the dev set.

预训练数据

3.2 Model Extensions

We improve our method by proposing and evaluating several extensions to the model. Unless stated otherwise, these experiments use the same model size and training data as BERT-Base.

Weight Sharing We propose improving the efficiency of the pre-training by sharing weights between the generator and discriminator. If the generator and discriminator are the same size, all of the encoder weights can be tied. However, we found it to be more efficient to have a small generator, in which case we only share the token embeddings of the generator and discriminator. In this case we use token embeddings the size of the discriminator's hidden states.[3] The "input" and "output" embeddings of the generator are always tied as in BERT.

We compare the weight tying strategies when the generator is the same size as the discriminator. We train these models for 500k steps. GLUE scores are 83.6 for no weight tying, 84.3 for tying token embeddings, and 84.4 for tying all weights. We hypothesize that ELECTRA benefits from tied token embeddings because masked language modeling is particularly effective at learning these representations: while the discriminator only updates tokens that are present in the input or are sampled by the generator, the generator's softmax over the vocabulary densely updates all token embeddings. On the other hand, tying all encoder weights caused little improvement while incurring the significant disadvantage of requiring the generator and discriminator to be the same size. Based on these findings, we use tied embeddings for further experiments in this paper.

Smaller Generators If the generator and discriminator are the same size, training ELECTRA would take around twice as much compute per step as training only with masked language modeling. We suggest using a smaller generator to reduce this factor. Specifically, we make models smaller by decreasing the layer sizes while keeping the other hyperparameters constant. We also explore using an extremely simple "unigram" generator that samples fake tokens according their frequency in the train corpus. GLUE scores for differently-sized generators and discriminators are shown in the left of Figure 3. All models are trained for 500k steps, which puts the smaller generators at a disadvantage in terms of compute because they require less compute per training step. Nevertheless, we find that models work best with generators 1/4-1/2 the size of the discriminator. We speculate that having too strong of a generator may pose a too-challenging task for the discriminator, preventing it from learning as effectively. Further experiments in this paper use the best generator size found for the given discriminator size.

Training Algorithms Lastly, we propose other training algorithms for ELECTRA, although these did not end up improving results. The proposed training objective jointly trains the generator and discriminator. We experiment with instead using the following two-stage training procedure:

[3]We add linear layers to the generator to project the embeddings into generator-hidden-sized representations.
控制输入进生成器的维度，进而控制生成器的规模

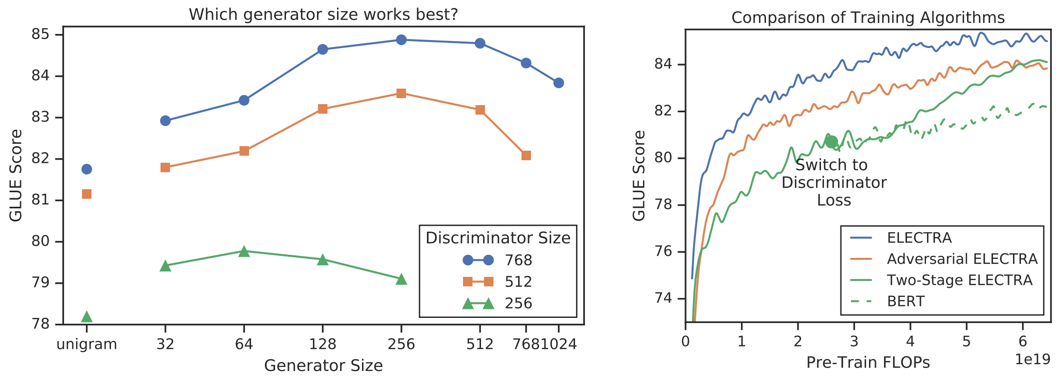

Figure 3: Left: GLUE scores for different generator/discriminator sizes (number of hidden units). Interestingly, having a generator smaller than the discriminator improves results. Right: Comparison of different training algorithms. As our focus is on efficiency, the x-axis shows FLOPs rather than train steps (e.g., ELECTRA is trained for fewer steps than BERT because it includes the generator).

训练步骤

1. Train only the generator with $\mathcal{L}_{\text{MLM}}$ for n steps.
2. Initialize the weights of the discriminator with the weights of the generator. Then train the discriminator with $\mathcal{L}_{\text{Disc}}$ for n steps, keeping the generator's weights frozen.

Note that the weight initialization in this procedure requires having the same size for the generator and discriminator. We found that without the weight initialization the discriminator would sometimes fail to learn at all beyond the majority class, perhaps because the generator started so far ahead of the discriminator. Joint training on the other hand naturally provides a curriculum for the discriminator where the generator starts off weak but gets better throughout training. We also explored training the generator adversarially as in a GAN, using reinforcement learning to accommodate the discrete operations of sampling from the generator. See Appendix D for details.

Results are shown in the right of Figure 3. During two-stage training, downstream task performance notably improves after the switch from the generative to discriminative the objective, but does not end up outperforming joint training. Although still outperforming BERT, we found adversarial training to underperform maximum-likelihood training. This negative result is not too surprising given the relatively poor performance of GANs for text generation in general (Caccia et al., 2018).

3.3 Small Models

As a goal of this work is to improve the efficiency of pre-training, we develop a small model that can be quickly trained on a single GPU. Starting with the BERT-Base hyperparameters, we shortened the sequence length (from 512 to 128), reduced the batch size (from 256 to 128), reduced the model's hidden dimension size (from 768 to 256), and used smaller token embeddings (from 768 to 128). To provide a fair comparison, we also train a BERT-Small model using the same hyperparameters. We train BERT-Small for 1.5M steps, so it uses the same training FLOPs as ELECTRA-Small, which was trained for 1M steps.[4] In addition to BERT, we compare against two less resource-intensive pre-training methods based on language modeling: ELMo (Peters et al., 2018) and GPT (Radford et al., 2018).[5] We also compare with DistilBERT,[6] a smaller Transformer distilled from BERT-Base.

Results are shown in Table 1. ELECTRA-Small performs remarkably well given its size, achieving a higher GLUE score than other methods using substantially more compute and parameters. The results also demonstrate the strength of ELECTRA at a more moderate size; our base-sized ELECTRA model substantially outperforms BERT-Base and even outperforms BERT-Large (which gets 84.0 GLUE score). We hope ELECTRA's ability to achieve strong results with relatively little compute will broaden the accessibility of developing and applying pre-trained models in NLP.

[4]ELECTRA requires more FLOPs per step because it consists of the generator as well as the discriminator.
[5]GPT is similar in size to BERT-Base, but is trained for fewer steps.
[6]`https://medium.com/huggingface/distilbert-8cf3380435b5`

Model	**Train / Infer FLOPs**	**Speedup**	**Params**	**Train Time + Hardware**	**GLUE**
ELMo	3.3e18 / 2.6e10	19x / 1.2x	96M	14d on 3 GTX 1080 GPUs	71.2
GPT	4.0e19 / 3.0e10	1.6x / 0.97x	117M	25d on 8 P6000 GPUs	78.8
BERT-Small	1.4e18 / 3.7e9	45x / 8x	14M	4d on 1 V100 GPU	74.7
BERT-Base	6.4e19 / 2.9e10	1x / 1x	110M	4d on 16 TPUv3s	82.2
DistilBERT	– / 1.4e10	– / 2x	66M	–	77.8
ELECTRA-Small	1.4e18 / 3.7e9	45x / 8x	14M	4D on 1 V100 GPU	79.0
ELECTRA-Base	6.4e19 / 2.9e10	1x / 1x	110M	4D on 16 TPUv3s	85.1

Table 1: Comparison of small models on the GLUE dev set. BERT-Small/Base are our implementation and use the same hyperparameters as ELECTRA-Small/Base. Infer FLOPs assumes single length-128 input. Training times should be taken with a grain of salt as they are for different hardware and with sometimes un-optimized code.

Model	**Train FLOPs**	**Params**	**CoLA**	**SST**	**MRPC**	**STS**	**QQP**	**MNLI**	**QNLI**	**RTE**	**Avg.**
BERT	1.9e20 (0.27x)	335M	60.6	93.2	88.0	90.0	91.3	86.6	92.3	70.4	84.0
XLNet	9.6e20 (1.3x)	360M	63.6	95.6	89.2	91.8	91.8	89.8	93.9	83.8	87.4
RoBERTa-100K	6.4e20 (0.90x)	356M	66.1	95.6	**91.4**	92.2	92.0	89.3	94.0	82.7	87.9
RoBERTa	3.2e21 (4.5x)	356M	68.0	**96.4**	90.9	**92.4**	92.2	90.2	**94.7**	86.6	88.9
BERT (ours)	7.1e20 (1x)	335M	67.0	95.9	89.1	91.2	91.5	89.6	93.5	79.5	87.2
ELECTRA	7.1e20 (1x)	335M	**69.3**	96.0	90.6	92.1	**92.4**	**90.5**	94.5	**86.8**	**89.0**
Test set results for models with standard single-task finetuning (no ensembling, task-specific tricks, etc.)											
BERT	1.9e20 (0.27x)	335M	60.5	94.9	89.3	86.5	89.3	86.7	92.7	70.1	83.8
SpanBERT	7.1e20 (1x)	335M	64.3	94.8	**90.9**	89.9	89.5	87.7	94.3	79.0	86.3
ELECTRA	7.1e20 (1x)	335M	**68.2**	**96.9**	89.6	**91.0**	**90.1**	**90.1**	**95.4**	**83.6**	**88.1**

Table 2: Comparison of large models on the GLUE dev and test sets. BERT dev results are from Clark et al. (2019). See Appendix B for some discussion on GLUE test set comparisons.

3.4 Large Models

We train big ELECTRA models to measure the effectiveness of ELECTRA at the large scale of current state-of-the-art pre-trained Transformers. Our ELECTRA-Large model is the same size as BERT-Large but is trained for much longer: approximately 400k steps with batch size 2048 on the XLNet data. We note that although the XLNet data is similar to the data used to train RoBERTa, the comparison isn't entirely direct. As a baseline, we trained our own BERT-Large model using the same hyperparameters and training time as ELECTRA-Large.

Results on GLUE are shown in Table 2. ELECTRA matches the performance of RoBERTa, the current state-of-the-art pre-trained transformer. However, it took less than 1/4 of the compute to train ELECTRA-Large as it did to train RoBERTa, demonstrating that ELECTRA's sample-efficiency gains hold at large scale. We believe ELECTRA would perform even better if trained longer. ELECTRA substantially outperforms RoBERTa trained for 100K steps and XLNet, which use similar amounts of compute as ELECTRA. Surprisingly, our baseline BERT model performs worse than RoBERTa-100K, suggesting our models may benefit from more hyperparameter tuning or using the RoBERTa training data. ELECTRA's large gain over BERT holds on the GLUE test set. Note that we did not use the performance-enhancing tricks employed by other recent GLUE leaderboard submissions (see Appendix B), so these test-set numbers are not comparable to those of RoBERTa/XLNet.

3.5 Efficiency Analysis

We have suggested that posing the training objective over a small subset of tokens makes masked language modeling inefficient. However, it isn't entirely obvious that this is the case. After all, the model still receives a large number of input tokens even though it to predict only a small number of masked tokens. To better understand where the gains from ELECTRA are coming from, we compare a series of other pre-training objectives:

Model	ELECTRA	All-Tokens MLM	Replace MLM	ELECTRA 15%	BERT
GLUE score	85.0	84.3	82.4	82.4	82.2

Table 3: Compute-efficiency experiments (see text for details).

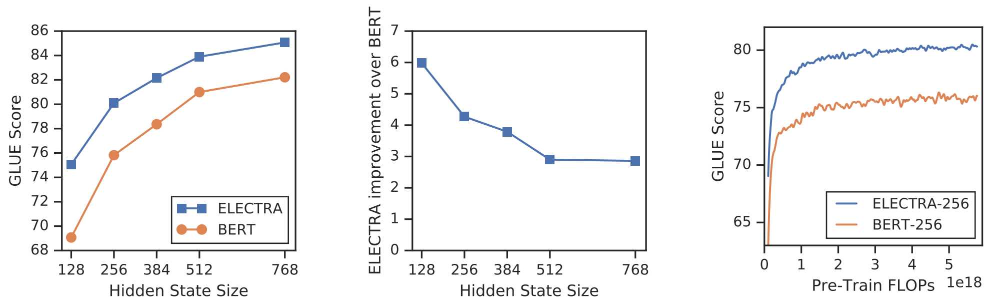

Figure 4: Left and Center: Comparison of BERT and ELECTRA for different model sizes. Right: A small ELECTRA model converges to higher downstream accuracy than BERT, showing the improvement comes from more than just faster training.

- **ELECTRA 15%**: This model is identical to ELECTRA except the discriminator loss only comes from the 15% of the tokens that were masked out of the input. In other words, the sum in the discriminator loss $\mathcal{L}_{\text{Disc}}$ is over $i \in \boldsymbol{m}$ instead of from 1 to n.[7]
- **Replace MLM**: This objective is the same as masked language modeling except instead of replacing masked-out tokens with `[MASK]`, they are replaced with tokens from a generator model. This objective tests to what extent ELECTRA's gains come from solving the discrepancy of exposing the model to `[MASK]` tokens during pre-training but not fine-tuning.
- **All-Tokens MLM**: Like in Replace MLM, masked tokens are replaced with generator samples. Furthermore, the model predicts the identity of all tokens in the input, not just ones that were masked out. We found it improved results to train this model with an explicit copy mechanism that outputs a copy probability D for each token using a sigmoid layer. The model's output distribution puts D weight on the input token plus $1 - D$ times the output of the MLM softmax. This model is essentially a combination of BERT and ELECTRA. Note that without generator replacements, the model would trivially learn to make predictions from the vocabulary for `[MASK]` tokens and copy the input for other ones.

Results are shown in Table 3. First, we find that ELECTRA is greatly benefiting from having a loss defined over all input tokens rather than just a subset: ELECTRA 15% performs much worse than ELECTRA. Secondly, we find that BERT performance is being slightly harmed from the pre-train fine-tune mismatch from `[MASK]` tokens, as Replace MLM slightly outperforms BERT. We note that BERT (including our implementation) already includes a trick to help with the pre-train/fine-tune discrepancy: masked tokens are replaced with a random token 10% of the time and are kept the same 10% of the time. However, our results suggest these simple heuristics are insufficient to fully solve the issue. Lastly, we find that All-Tokens MLM, the generative model that makes predictions over all tokens instead of a subset, closes most of the gap between BERT and ELECTRA. In total, these results suggest a large amount of ELECTRA's improvement can be attributed to learning from all tokens and a smaller amount can be attributed to alleviating the pre-train fine-tune mismatch.

The improvement of ELECTRA over All-Tokens MLM suggests that the ELECTRA's gains come from more than just faster training. We study this further by comparing BERT to ELECTRA for various model sizes (see Figure 4). We find that the gains from ELECTRA grow larger as the models get smaller. The small models are trained fully to convergence, showing that ELECTRA achieves higher downstream accuracy than BERT when fully trained. We speculate that ELECTRA

[7]We also trained a discriminator that learns from a random 15% of the input tokens distinct from the subset that was originally masked out. This model performed slightly worse.

is more parameter-efficient than BERT because it is trained as a discriminative classifier and so does not have to model the full data distribution.

4 RELATED WORK

Self-Supervised Pre-training for NLP Self-supervised learning has been used to learn word representations (Collobert et al., 2011; Pennington et al., 2014) and more recently *contextual* representations of words though objectives such as language modeling (Dai & Le, 2015; Peters et al., 2018; Howard & Ruder, 2018). BERT (Devlin et al., 2019) pre-trains a large Transformer (Vaswani et al., 2017) at the masked-language modeling task. There have been numerous extensions to BERT. For example, MASS (Song et al., 2019) and UniLM (Dong et al., 2019) extend BERT to generation tasks by adding auto-regressive generative training objectives. ERNIE (Sun et al., 2019) and SpanBERT (Joshi et al., 2019) mask out contiguous sequences of token for improved span representations. Instead of masking out input tokens, XLNet (Yang et al., 2019) masks attention weights such that the input sequence is auto-regressively generated in a random order. However, this method suffers from the same inefficiencies as BERT because XLNet only generates 15% of the input tokens in this way. Like ELECTRA, XLNet may alleviate BERT's pretrain-finetune discrepancy by not requiring `[MASK]` tokens, although this isn't entirely clear because XLNet uses two "streams" of attention during pre-training but only one for fine-tuning.

Generative Adversarial Networks GANs (Goodfellow et al., 2014) are effective at generating high-quality synthetic data. Radford et al. (2016) propose using the discriminator of a GAN in downstream tasks, which is similar to our method. Large-scale adversarial learning (Brock et al., 2019) is currently state-of-the-art for unsupervised pre-training of image classifiers, albeit with a different approach based on BiGAN (Donahue et al., 2017). GANs have been applied to text data (Yu et al., 2017; Zhang et al., 2017), although state-of-the-art approaches still lag behind standard maximum-likelihood training (Caccia et al., 2018; Tevet et al., 2018). Although we do not use adversarial learning, our generator is particularly reminiscent of MaskGAN (Fedus et al., 2018), which trains the generator to fill in tokens deleted from the input.

Contrastive Learning Broadly speaking, contrastive learning methods distinguish observed data points from fictitious negative samples. They have been applied to many modalities including text (Smith & Eisner, 2005), images (Chopra et al., 2005), and video (Wang & Gupta, 2015; Sermanet et al., 2017) data. Common approaches learn embedding spaces where related data points are similar (Saunshi et al., 2019) or models that rank real data points over negative samples (Collobert et al., 2011; Bordes et al., 2013). Noise-Contrastive Estimation (Gutmann & Hyvärinen, 2010) and Contrastive Predictive Encoding (van den Oord et al., 2018) train a generative model to score real data points higher than negative samples. Our method is a bit different in that it it trains a *discriminative* model to distinguish real and fictitious tokens.

Word2Vec (Mikolov et al., 2013), one of the earliest pre-training methods for NLP, uses contrastive learning. In fact, ELECTRA can be viewed as a massively scaled-up version of Continuous Bag-of-Words (CBOW) with Negative Sampling. CBOW also predicts an input token given surrounding context and negative sampling rephrases the learning task as a binary classification task on whether the input token comes from the data or proposal distribution. However, CBOW uses a bag-of-vectors encoder rather than a transformer and a simple proposal distribution derived from unigram token frequencies instead of a learned generator.

5 CONCLUSION

We have proposed replaced token detection, a new self-supervised task for language representation learning. The key idea is training a text encoder to distinguish input tokens from high-quality negative samples produced by an small generator network. Compared to masked language modeling, our pre-training objective is more compute-efficient and results in better performance on downstream tasks. It works well even when using relatively small amounts of compute, which we hope will make developing and applying pre-trained text encoders more accessible to researchers and practitioners with less access to computing resources. We also hope more future work on NLP pre-training will consider efficiency as well as absolute performance, and follow our effort in reporting compute usage and parameter counts along with evaluation metrics.

REFERENCES

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.

Avishek Joey Bose, Huan Ling, and Yanshuai Cao. Adversarial contrastive estimation. In *ACL*, 2018.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019.

Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, and Laurent Charlin. Language GANs falling short. *arXiv preprint arXiv*, 2018.

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

Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. *CVPR*, 2005.

Kevin Clark, Minh-Thang Luong, Urvashi Khandelwal, Christopher D. Manning, and Quoc V. Le. BAM! Born-again multi-task networks for natural language understanding. In *ACL*, 2019.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *JMLR*, 2011.

Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *NIPS*, 2015.

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

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. In *ICLR*, 2017.

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

William Fedus, Ian J. Goodfellow, and Andrew M. Dai. MaskGAN: Better text generation via filling in the _______. In *ICLR*, 2018.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS*, 2010.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *ACL*, 2018.

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

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

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

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *ICLR Workshop Papers*, 2013.

Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. English gigaword fifth edition, linguistic data consortium. 2011.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

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

Jason Phang, Thibault Févry, and Samuel R Bowman. Sentence encoders on STILTs: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*, 2018.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *https://blog.openai.com/language-unsupervised*, 2018.

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

Nikunj Saunshi, Orestis Plevrakis, Sanjeev Arora, Mikhail Khodak, and Hrishikesh Khandeparkar. A theoretical analysis of contrastive unsupervised representation learning. In *ICML*, 2019.

Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, and Sergey Levine. Time-contrastive networks: Self-supervised learning from video. *ICRA*, 2017.

Noah A. Smith and Jason Eisner. Contrastive estimation: Training log-linear models on unlabeled data. In *ACL*, 2005.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. MASS: Masked sequence to sequence pre-training for language generation. *arXiv preprint arXiv:1905.02450*, 2019.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. Ernie: Enhanced representation through knowledge integration. *arXiv preprint arXiv:1904.09223*, 2019.

Guy Tevet, Gavriel Habib, Vered Shwartz, and Jonathan Berant. Evaluating text gans as language models. In *NAACL-HLT*, 2018.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

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

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008.

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

Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. *ICCV*, pp. 2794–2802, 2015.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. XLNet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

Yang You, Jing Li, Jonathan Hseu, Xiaodan Song, James Demmel, and Cho-Jui Hsieh. Reducing bert pre-training time from 3 days to 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

Lantao Yu, Weinan Zhang, Jun Wang, and Yingrui Yu. SeqGAN: Sequence generative adversarial nets with policy gradient. In *AAAI*, 2017.

Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. Adversarial feature matching for text generation. In *ICML*, 2017.

Hyperparameter	Small	Base	Large
Number of layers	12	12	24
Hidden Size	256	768	1024
FFN inner hidden size	1024	3072	4096
Attention heads	4	12	16
Attention head size	64	64	64
Embedding Size	128	768	1024
Generator Size (multiplier for hidden-size, FFN-size, and num-attention-heads)	1/2	1/3	1/4
Learning Rate Decay	Linear	Linear	Linear
Warmup steps	10000	10000	10000
Learning Rate	5e-4	2e-4	2e-4
LAMB ϵ	1e-6	1e-6	1e-6
LAMB β_1	0.9	0.9	0.9
LAMB 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	128	256	2048
Train Steps (BERT/ELECTRA)	1.45M/1M	1M/766K	464K/400K

Table 4: Pre-train hyperparameters

Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *ICCV*, 2015.

A Pre-Training Details

The following details apply to both our BERT and ELECTRA models. We mostly use the same hyperparameters as BERT. However, one difference is that we use the LAMB (You et al., 2019) optimizer instead of Adam (Kingma & Ba, 2015). While it did not outperform Adam when Adam was well-tuned, it required less tuning over differing train batch sizes, loss functions, etc. We set λ, the weight for the discriminator objective in the loss to 50.[8] We use dynamic token masking with the masked positions decided on-the-fly instead of during preprocessing. Also, we did not use the next sentence prediction objective proposed in the original BERT paper, as recent work has suggested it does to improve performance (Yang et al., 2019; Liu et al., 2019). We searched for the best learning rate for the Base and Small models out of [1e-4, 2e-4, 3e-4, 5e-4] and selected λ out of [1, 10, 20, 50, 100] in early experiments. Otherwise we did no hyperparameter tuning beyond the experiments in Section 3.2. The full set of hyperparameters are listed in Table 4. Training ELECTRA-Large took 5 days on 256 TPUv3s; other train times and hardware are listed in Table 1.

B Fine-Tuning Details

For Large-sized models, we used the hyperparameters from Clark et al. (2019) for the most part (including the use of Adam). However, after noticing that RoBERTa (Liu et al., 2019) uses more training epochs (10 rather than 3) we searched for the best number of train epochs out of [10, 3] for each task. For Base-sized models we searched for a learning rate out of [3e-5, 5e-5, 1e-4, 1.5e-4] and the layer-wise learning-rate decay out of [0.9, 0.8, 0.7], but otherwise used the same hyperparameters as for Large models. We kept these hyperparameters the same for small models. With the exception of number of train epochs, we used the same hyperparameters for all tasks rather

[8] As a binary classification task instead of the 30,000-way classification task in MLM, the discriminator's loss was typically much lower than the generator's

Hyperparameter	Value
Learning Rate	1e-4 for Base/Small, 5e-5 for Large
Adam ϵ	1e-6
Adam β_1	0.9
Adam β_2	0.999
Layerwise LR decay	0.8 for Base/Small, 0.9 for Large
Learning rate decay	Linear
Warmup fraction	0.1
Attention Dropout	0.1
Dropout	0.1
Weight Decay	0
Batch Size	32
Train Epochs	10 for RTE and STS, 3 for other tasks

Table 5: Fine-tune hyperparameters

than separately searching for the best hyperparameters for each one individually as most prior work has done. The full set of hyperparameters is listed in Table 5.

Following BERT, we do not show results on the WNLI GLUE task, as it is difficult to beat even the majority classifier using a standard fine-tuning-as-classifier approach. We report most results on the GLUE dev set. One reason for this is that results vary significantly (standard deviation of around ± 0.4 GLUE points) between fine-tuning runs with different random seeds, so we think reporting the median score over multiple fine-tuning runs is very important. However, test results are only obtainable by a leaderboard submission for a single training run, making comparisons less reliable. Furthermore, it is difficult to have apples-to-apples comparisons with prior work on the test sets due to the large number of tricks commonly employed to increase leaderboard scores. Top GLUE leaderboard entries are ensembles, use two-stage STILTS (Phang et al., 2018) training for some tasks, and apply task-specific tricks for some tasks (e.g., treating QNLI as a ranking task). As different approaches use different tricks tuned different ways, we decided to instead use a simple single-task non-ensemble fine-tuning approach (the same as BERT) for our test set submission and compare with other methods that provide results in the same setting.

C Counting FLOPs

We used TensorFlow's FLOPs-counting capabilities[9] and checked the results with by-hand computation. We made the following assumptions:

- An "operation" is a mathematical operation, not a machine instruction. For example, an `exp` is one op like an `add`, even though in practice the `exp` might be slower. We believe this assumption does not substantially change compute estimates because matrix-multiplies dominate the compute for most models. Similarly, we count matrix-multiplies as $2 * m * n$ FLOPs instead of $m * n$ as one might if considering fused multiply-add operations.
- The backwards pass takes the same number of FLOPs as the forward pass. This assumption is not exactly right (e.g, for softmax cross entropy loss the backward pass is faster). Importantly, the forward/backward pass FLOPs really is the same for matrix-multiplies, which is most of the compute anyway.
- We assume "dense" embedding lookups (i.e., multiplication by a one-hot vector). In practice, sparse embedding lookups are much slower than constant time; on some hardware accelerators dense operations are actually faster than sparse lookups.

[9]See `https://www.tensorflow.org/api_docs/python/tf/profiler`

D Adversarial Training

Here we detail attempts to adversarially train the generator instead of using maximum likelihood. In particular we train the generator G to maximize the discriminator loss $\mathcal{L}_{\text{Disc}}$. As our discriminator isn't precisely the same as the discriminator of a GAN (see the discussion in Section 2), this method is really an instance of Adversarial Contrastive Estimation (Bose et al., 2018) rather than Generative Adversarial Training.

Our generator is different from most text generation models in that it is non-autogregressive: predictions are made independently. In other words, rather than taking a sequence of actions where each action generates a token, the generator takes a single giant action of generating all tokens simultaneously, where the probability for the action factorizes as the product of generator probabilities for each token. To deal with this enormous action space, we make the following simplifying assumption: that the discriminator's prediction $D(\boldsymbol{x}^{\text{corrupt}}, t)$ depends only on the token x_t and the non-replaced tokens $\{x_i : i \notin \boldsymbol{m}\}$, i.e., it does not depend on generated tokens $\{\hat{x}_i : i \in \boldsymbol{m} \wedge i \neq t\}$. This isn't too bad of an assumption to make because a relatively small tokens are replaced. Notationally, we show this assumption by (in a slight abuse of notation) by writing $D(\hat{x}_t|\boldsymbol{x}^{\text{masked}})$ for the discriminator predicting whether the generated token $\hat{x}_t$ equals the original token x_t given the masked context $\boldsymbol{x}^{\text{masked}}$. A useful consequence of this assumption is that the discriminator score for non-replaced tokens ($D(x_t|\boldsymbol{x}^{\text{masked}})$ for $t \notin \boldsymbol{m}$) is independent of p_G because it does not depend on any replaced token. Therefore it can be ignored when training G to maximize the $\mathcal{L}_{\text{Disc}}$. During training we seek to find

$$\arg\max_{\theta_G} \mathcal{L}_{\text{Disc}} = \arg\max_{\theta_G} \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{m},\hat{\boldsymbol{x}}} \Bigg(\sum_{t=1}^{n} \mathbb{1}(x_t^{\text{corrupt}} = x_t) \log D(\boldsymbol{x}^{\text{corrupt}}, t) + \\ \mathbb{1}(x_t^{\text{corrupt}} \neq x_t) \log(1 - D(\boldsymbol{x}^{\text{corrupt}}, t)) \Bigg)$$

Using the simplifying assumption, we approximate the above by finding the argmax of

$$\mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{m},\hat{\boldsymbol{x}}} \Bigg(\sum_{t \in \boldsymbol{m}} \mathbb{1}(\hat{x}_t = x_t) \log D(\hat{x}|\boldsymbol{x}^{\text{masked}}) + \mathbb{1}(\hat{x}_t \neq x_t) \log(1 - D(\hat{x}|\boldsymbol{x}^{\text{masked}})) \Bigg)$$

$$= \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{m}} \sum_{t \in \boldsymbol{m}} \mathop{\mathbb{E}}_{\hat{x}_t \sim p_G} R(\hat{x}_t, \boldsymbol{x})$$

$$\text{where } R(x_i', \boldsymbol{x}) = \begin{cases} \log D(\hat{x}|\boldsymbol{x}^{\text{masked}}) & \text{if } \hat{x}_t = x_t \\ \log(1 - D(\hat{x}|\boldsymbol{x}^{\text{masked}})) & \text{otherwise} \end{cases}$$

In short, the simplifying assumption allows us to decompose the loss over the individual generator samples. We cannot directly find $\arg\max_{\theta_G}$ using gradient ascent because it is impossible to backpropagate through discrete sampling of $\hat{x}$. Instead, we use policy gradient reinforcement learning (Williams, 1992). In particular, we use the REINFORCE gradient

$$\nabla_{\theta_G} \mathcal{L}_{\text{Disc}} \approx \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{m}} \sum_{t \in \boldsymbol{m}} \mathop{\mathbb{E}}_{\hat{x}_t \sim p_G} \nabla_{\theta_g} \log p_G(\hat{x}_t|\boldsymbol{x}^{\text{masked}})[R(\hat{x}_t, \boldsymbol{x}) - b(\boldsymbol{x}^{\text{masked}}, t)]$$

Where b is a learned baseline implemented as $b(\boldsymbol{x}^{\text{masked}}, t) = \log \text{sigmoid}(w^T h_G(\boldsymbol{x}^{\text{masked}})_t)$ where $h_G(\boldsymbol{x}^{\text{masked}})$ are the outputs of the generator's Transformer encoder. The baseline is trained with cross-entropy loss to match the for the corresponding position. We approximate the expectations with a single sample and learn θ_G with gradient ascent. Despite receiving no explicit feedback about which generated tokens are correct, we found the adversarial training resulted in a fairly accurate generator (for a 256-hidden-size generator, the adversarially trained one achieves 54% accuracy at masked language modeling while the same sized MLE generator gets 65%). However, using this generator did not improve over the MLE-trained one on downstream tasks (see the right of Figure 3 in the main paper).

E Evaluating ELECTRA as a Masked Language Model

This sections details some initial experiments in evaluating ELECTRA as a masked language model. Using slightly different notation from the main paper, given a context c consisting of a text sequence with one token masked-out, the discriminator loss can be written as.

$$\begin{aligned}\mathcal{L}_{\text{Disc}} = \sum_{x \in \text{vocab}} \Big(& (1 - p_{\text{mask}}) p_{\text{data}}(x|c) \log D(x, c) + && \text{//unmasked token} \\ & p_{\text{mask}} p_{\text{data}}(x|c) p_G(x|c) \log D(x, c) + && \text{//generator samples correct token} \\ & p_{\text{mask}}(1 - p_{\text{data}}(x|c)) p_G(x|c) \log(1 - D(x, c)) \Big) && \text{//generator samples incorrect token}\end{aligned}$$

Finding the critical points of this loss with respect to D shows that for a fixed generator the optimal discriminator is

$$D(x, c) = p_{\text{data}}(x|c)(a + p_G(x|c))/(ap_{\text{data}}(x|c) + p_G(x|c)$$

which means

$$p_{\text{data}}(x|c) = D(x, c)p_G(x|c)/(a(1 - D(x, c)) + p_G(x|c))$$

where $a = (1 - p_{\text{mask}})/p_{\text{mask}}$ is the number of unmasked tokens for every masked token. We can use this expression to evaluate ELECTRA as a masked language model by selecting $\text{argmax}_{x \in \text{vocab}} D(x, c)p_G(x|c)/(a(1 - D(x, c)) + p_G(x|c))$ as the model's prediction for a given context. In practice, selecting over the whole vocabulary is very expensive, so we instead take the argmax over the top 100 predictions from the generator.[10] Using this method, we compared ELECTRA-Base and BERT-Base on the Wikipedia+BooksCorpus dataset. We found that BERT slightly outperformed ELECTRA at masked language modeling (77.9% vs 75.5% accuracy). It is possible that the assumption of an optimal discriminator, which is certainly far from correct, is harming ELECTRA's accuracy under this evaluation scheme. However, perhaps it is not too surprising that a model like BERT that is trained specifically for generation performs better at generation while a model with a discriminative objective like ELECTRA is better at being fine-tuned on discriminative tasks. We think comparisons of BERT's and ELECTRA's MLM predictions might be an interesting way to uncover more about the differences between ELECTRA and BERT encoders in future work.

F Negative Results

We briefly describe a few ideas that did not look promising in our initial experiments:

- We initially attempted to make BERT more efficient by strategically masking-out tokens (e.g., masking our rarer tokens more frequently, or training a model to guess which tokens BERT would struggle to predict if they were masked out). This resulted in fairly minor speedups over regular BERT.
- Given that ELECTRA seemed to benefit (up to a certain point) from having a weaker generator (see Section 3.2), we explored raising the temperature of the generator's output softmax or disallowing the generator from sampling the correct token. Neither of these improved results.
- We tried adding a sentence-level contrastive objective. For this task, we kept 20% of input sentences unchanged rather than noising them with the generator. We then trained added a prediction head to the model that predicted if the entire input was corrupted or not. Surprisingly, this slightly decreased performance on downstream tasks.

[10]for ELECTRA-Base this means the upper-bound for accuracy is around 95%