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

**Kevin Clark**
Stanford University
kevclark@cs.stanford.edu

**Minh-Thang Luong**
Google Brain
thangluong@google.com

**Quoc V. Le**
Google Brain
qvl@google.com

**Christopher D. Manning**
Stanford University & CIFAR Fellow
manning@cs.stanford.edu

arXiv:2003.10555v1 [cs.CL] 23 Mar 2020

## ABSTRACT

Masked language modeling (MLM) pre-training methods such as BERT corrupt the input by replacing some tokens with [MASK] and then train a model to reconstruct the original tokens. While they produce good results when transferred to downstream NLP tasks, they generally require large amounts of compute to be effective. 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 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 task is defined over *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 BERT 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 it performs comparably to RoBERTa and XLNet while using less than 1/4 of their compute and outperforms them when using the same amount of compute.

## 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 (e.g., BERT; Devlin et al. (2019)) or attention to those tokens (e.g., XLNet; Yang et al. (2019)), and then train the network to recover the original input. While more effective than conventional language-model pre-training due to learning bidirectional representations, these masked language modeling (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 real input tokens from plausible but synthetically generated replacements. 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. A key 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

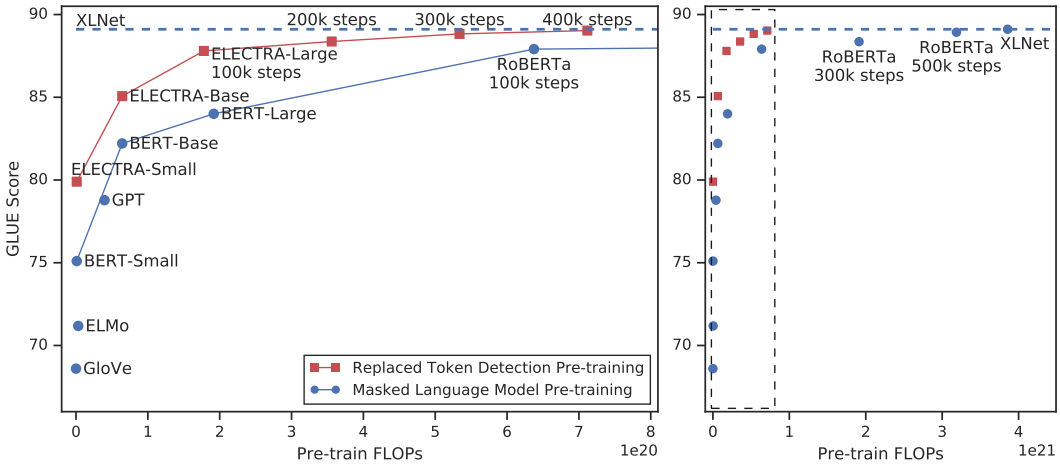


Figure 1: Replaced token detection pre-training consistently outperforms masked language model pre-training given the same compute budget. The left figure is a zoomed-in view of the dashed box.

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).

We call our approach ELECTRA[1] for "Efficiently Learning an Encoder that Classifies Token Replacements Accurately." As in prior work, we apply it to pre-train Transformer text encoders (Vaswani et al., 2017) that can be fine-tuned on downstream tasks. Through a series of ablations, we show that learning from all input positions 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 downstream performance vs. their compute requirement. In particular, we run experiments on the GLUE natural language understanding benchmark (Wang et al., 2019) and SQuAD question answering benchmark (Rajpurkar et al., 2016). ELECTRA substantially 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.[2] 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 an ELECTRA-Large model that performs comparably to RoBERTa (Liu et al., 2019) and XLNet (Yang et al., 2019), despite having fewer parameters and using 1/4 of the compute for training. Training ELECTRA-Large further results in an even stronger model that outperforms ALBERT (Lan et al., 2019) on GLUE and sets a new state-of-the-art for SQuAD 2.0. 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.

[1]Code and pre-trained weights will be released at `https://github.com/google-research/electra`

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

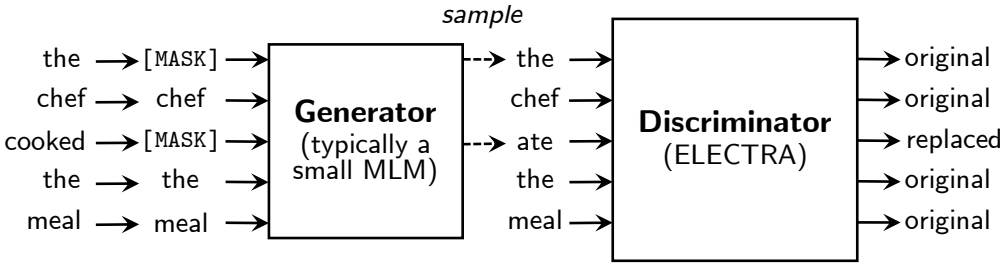


Figure 2: An overview of replaced token detection. The generator can be any model that produces an output distribution over tokens, but we usually use a small masked language model that is trained jointly with the discriminator. Although the models are structured like in a GAN, we train the generator 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 (the ELECTRA model) on downstream tasks.

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 $t$, (in our case only positions where $x_t = \texttt{[MASK]}$), the generator outputs a probability for generating a particular token $x_t$ with a softmax layer:

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

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

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

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 $\boldsymbol{m} = [m_1, ..., m_k]$.[3] 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 predict the original identities 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

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

and the loss functions are

$$\mathcal{L}_{\text{MLM}}(\boldsymbol{x}, \theta_G) = \mathbb{E}\left(\sum_{i \in \boldsymbol{m}} -\log p_G(x_i|\boldsymbol{x}^{\text{masked}})\right)$$

$$\mathcal{L}_{\text{Disc}}(\boldsymbol{x}, \theta_D) = \mathbb{E}\left(\sum_{t=1}^{n} -\mathbb{1}(x_t^{\text{corrupt}} = x_t) \log D(\boldsymbol{x}^{\text{corrupt}}, t) - \mathbb{1}(x_t^{\text{corrupt}} \neq x_t) \log(1 - D(\boldsymbol{x}^{\text{corrupt}}, t))\right)$$

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

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

by using reinforcement learning to train the generator (see Appendix F), 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

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

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.

# 3 Experiments

## 3.1 Experimental Setup

We evaluate on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019) and Stanford Question Answering (SQuAD) dataset (Rajpurkar et al., 2016). GLUE contains a variety of tasks covering textual entailment (RTE and MNLI) question-answer entailment (QNLI), paraphrase (MRPC), question paraphrase (QQP), textual similarity (STS), sentiment (SST), and linguistic acceptability (CoLA). See Appendix C for more details on the GLUE tasks. Our evaluation metrics are Spearman correlation for STS, Matthews correlation for CoLA, and accuracy for the other GLUE tasks; we generally report the average score over all tasks. For SQuAD, we evaluate on versions 1.1, in which models select the span of text answering a question, and 2.0, in which some questions are unanswerable by the passage. We use the standard evaluation metrics of Exact-Match (EM) and F1 scores. 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. For fine-tuning on GLUE, we add simple linear classifiers on top of ELECTRA. For SQuAD, we add the question-answering module from XLNet on top of ELECTRA, which is slightly more sophisticated than BERT's in that it jointly rather than independently predicts the start and end positions and has a "answerability" classifier added for SQuAD 2.0. 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. See the appendix for further training details and hyperparameter values.

## 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 transformer weights can be tied. However, we found it to be more efficient to have a small generator, in which case we only share the embeddings (both the token and positional embeddings) of the generator and discriminator. In this case we use embeddings the size of the discriminator's hidden states.[4] The "input" and "output" token 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

[4]We add linear layers to the generator to project the embeddings into generator-hidden-sized representations.

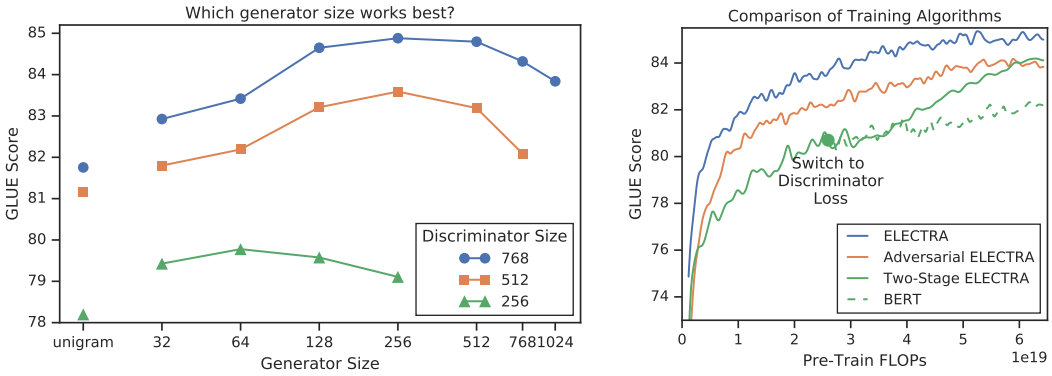


Figure 3: Left: GLUE scores for different generator/discriminator sizes (number of hidden units). Interestingly, having a generator smaller than the discriminator improves results. Right: Comparison of different training algorithms. As our focus is on efficiency, the x-axis shows FLOPs rather than train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

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. In particular, the discriminator may have to use many of its parameters modeling the generator rather than the actual data distribution. Further experiments in this paper use the best generator size found for the given discriminator size.

**Training Algorithms** Lastly, we explore 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:

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 F 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 the discriminative objective, but does not end up outscoring joint training. Although still outperforming BERT, we found adversarial training to underperform maximum-likelihood training. Further analysis suggests the gap is caused by two

| 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 | 75.1 |
| BERT-Base | 6.4e19 / 2.9e10 | 1x / 1x | 110M | 4d on 16 TPUv3s | 82.2 |
| ELECTRA-Small | 1.4e18 / 3.7e9 | 45x / 8x | 14M | 4d on 1 V100 GPU | 79.9 |
| 50% trained | 7.1e17 / 3.7e9 | 90x / 8x | 14M | 2d on 1 V100 GPU | 79.0 |
| 25% trained | 3.6e17 / 3.7e9 | 181x / 8x | 14M | 1d on 1 V100 GPU | 77.7 |
| 12.5% trained | 1.8e17 / 3.7e9 | 361x / 8x | 14M | 12h on 1 V100 GPU | 76.0 |
| 6.25% trained | 8.9e16 / 3.7e9 | 722x / 8x | 14M | 6h on 1 V100 GPU | 74.1 |
| 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. ELECTRA performs well even when trained on a single GPU, scoring 5 GLUE points higher than a comparable BERT model and even outscoring the much larger GPT model.

problems with adversarial training. First, the adversarial generator is simply worse at masked language modeling; it achieves 58% accuracy at masked language modeling compared to 65% accuracy for an MLE-trained one. We believe the worse accuracy is mainly due to the poor sample efficiency of reinforcement learning when working in the large action space of generating text. Secondly, the adversarially trained generator produces a low-entropy output distribution where most of the probability mass is on a single token, which means there is not much diversity in the generator samples. Both of these problems have been observed in GANs for text in prior work (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.[5] 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).[6] We also show results for a base-sized ELECTRA model comparable to BERT-Base.

Results are shown in Table 1. See Appendix D for additional results, including stronger small-sized and base-sized models trained with more compute. ELECTRA-Small performs remarkably well given its size, achieving a higher GLUE score than other methods using substantially more compute and parameters. For example, it scores 5 points higher than a comparable BERT-Small model and even outperforms the much larger GPT model. ELECTRA-Small is trained mostly to convergence, with models trained for even less time (as little as 6 hours) still achieving reasonable performance. While small models distilled from larger pre-trained transformers can also achieve good GLUE scores (Sun et al., 2019b; Jiao et al., 2019), these models require first expending substantial compute to pre-train the larger teacher model. The results also demonstrate the strength of ELECTRA at a 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.

[5] ELECTRA requires more FLOPs per step because it consists of the generator as well as the discriminator.

[6] GPT is similar in size to BERT-Base, but is trained for fewer steps.

| 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 |
| 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-500K | 3.2e21 (4.5x) | 356M | 68.0 | 96.4 | 90.9 | 92.1 | 92.2 | 90.2 | 94.7 | 86.6 | 88.9 |
| XLNet | 3.9e21 (5.4x) | 360M | 69.0 | **97.0** | 90.8 | 92.2 | 92.3 | 90.8 | 94.9 | 85.9 | 89.1 |
| 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-400K | 7.1e20 (1x) | 335M | **69.3** | 96.0 | 90.6 | 92.1 | **92.4** | 90.5 | 94.5 | 86.8 | 89.0 |
| ELECTRA-1.75M | 3.1e21 (4.4x) | 335M | 69.1 | 96.9 | 90.8 | **92.6** | **92.4** | **90.9** | **95.0** | **88.0** | **89.5** |

Table 2: Comparison of large models on the GLUE dev set. ELECTRA and RoBERTa are shown for different numbers of pre-training steps, indicated by the numbers after the dashes. ELECTRA performs comparably to XLNet and RoBERTa when using less than 1/4 of their pre-training compute and outperforms them when given a similar amount of pre-training compute. BERT dev results are from Clark et al. (2019).

| Model | Train FLOPs | CoLA | SST | MRPC | STS | QQP | MNLI | QNLI | RTE | WNLI | Avg.* | Score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BERT | 1.9e20 (0.06x) | 60.5 | 94.9 | 85.4 | 86.5 | 89.3 | 86.7 | 92.7 | 70.1 | 65.1 | 79.8 | 80.5 |
| RoBERTa | 3.2e21 (1.02x) | 67.8 | 96.7 | 89.8 | 91.9 | 90.2 | 90.8 | 95.4 | 88.2 | 89.0 | 88.1 | 88.1 |
| ALBERT | 3.1e22 (10x) | 69.1 | **97.1** | **91.2** | 92.0 | 90.5 | **91.3** | – | 89.2 | 91.8 | 89.0 | – |
| XLNet | 3.9e21 (1.26x) | 70.2 | **97.1** | 90.5 | **92.6** | 90.4 | 90.9 | – | 88.5 | **92.5** | 89.1 | – |
| ELECTRA | 3.1e21 (1x) | **71.7** | **97.1** | 90.7 | 92.5 | **90.8** | **91.3** | **95.8** | **89.8** | **92.5** | **89.5** | **89.4** |

Table 3: GLUE test-set results for large models. Models in this table incorporate additional tricks such as ensembling to improve scores (see Appendix B for details). Some models do not have QNLI scores because they treat QNLI as a ranking task, which has recently been disallowed by the GLUE benchmark. To compare against these models, we report the average score excluding QNLI (Avg.*) in addition to the GLUE leaderboard score (Score). "ELECTRA" and "RoBERTa" refer to the fully-trained ELECTRA-1.75M and RoBERTa-500K models.

### 3.4 Large Models

We train big ELECTRA models to measure the effectiveness of the replaced token detection pre-training task at the large scale of current state-of-the-art pre-trained Transformers. Our ELECTRA-Large models are the same size as BERT-Large but are trained for much longer. In particular, we train a model for 400k steps (ELECTRA-400K; roughly 1/4 the pre-training compute of RoBERTa) and one for 1.75M steps (ELECTRA-1.75M; similar compute to RoBERTa). We use a batch size 2048 and the XLNet pre-training data. We note that although the XLNet data is similar to the data used to train RoBERTa, the comparison is not entirely direct. As a baseline, we trained our own BERT-Large model using the same hyperparameters and training time as ELECTRA-400K.

Results on the GLUE dev set are shown in Table 2. ELECTRA-400K performs comparably to RoBERTa and XLNet. However, it took less than 1/4 of the compute to train ELECTRA-400K as it did to train RoBERTa and XLNet, demonstrating that ELECTRA's sample-efficiency gains hold at large scale. Training ELECTRA for longer (ELECTRA-1.75M) results in a model that outscores them on most GLUE tasks while still requiring less pre-training compute. Surprisingly, our baseline BERT model scores notably worse than RoBERTa-100K, suggesting our models may benefit from more hyperparameter tuning or using the RoBERTa training data. ELECTRA's gains hold on the GLUE test set (see Table 3), although these comparisons are less apples-to-apples due to the additional tricks employed by the models (see Appendix B).

Results on SQuAD are shown in Table 4. Consistent, with the GLUE results, ELECTRA scores better than masked-language-modeling-based methods given the same compute resources. For example, ELECTRA-400K outperforms RoBERTa-100k and our BERT baseline, which use similar amounts of pre-training compute. ELECTRA-400K also performs comparably to RoBERTa-500K despite using less than 1/4th of the compute. Unsurprisingly, training ELECTRA longer improves results further: ELECTRA-1.75M scores higher than previous models on the SQuAD 2.0 bench-

| Model | Train FLOPs | Params | SQuAD 1.1 dev | | SQuAD 2.0 dev | | SQuAD 2.0 test | |
|---|---|---|---|---|---|---|---|---|
| | | | EM | F1 | EM | F1 | EM | F1 |
| BERT-Base | 6.4e19 (0.09x) | 110M | 80.8 | 88.5 | – | – | – | – |
| BERT | 1.9e20 (0.27x) | 335M | 84.1 | 90.9 | 79.0 | 81.8 | 80.0 | 83.0 |
| SpanBERT | 7.1e20 (1x) | 335M | 88.8 | 94.6 | 85.7 | 88.7 | 85.7 | 88.7 |
| XLNet-Base | 6.6e19 (0.09x) | 117M | 81.3 | – | 78.5 | – | – | – |
| XLNet | 3.9e21 (5.4x) | 360M | **89.7** | **95.1** | 87.9 | **90.6** | 87.9 | 90.7 |
| RoBERTa-100K | 6.4e20 (0.90x) | 356M | – | 94.0 | – | 87.7 | – | – |
| RoBERTa-500K | 3.2e21 (4.5x) | 356M | 88.9 | 94.6 | 86.5 | 89.4 | 86.8 | 89.8 |
| ALBERT | 3.1e22 (44x) | 235M | 89.3 | 94.8 | 87.4 | 90.2 | 88.1 | 90.9 |
| BERT (ours) | 7.1e20 (1x) | 335M | 88.0 | 93.7 | 84.7 | 87.5 | – | – |
| ELECTRA-Base | 6.4e19 (0.09x) | 110M | 84.5 | 90.8 | 80.5 | 83.3 | – | – |
| ELECTRA-400K | 7.1e20 (1x) | 335M | 88.7 | 94.2 | 86.9 | 89.6 | – | – |
| ELECTRA-1.75M | 3.1e21 (4.4x) | 335M | **89.7** | 94.9 | **88.0** | **90.6** | **88.7** | **91.4** |

Table 4: Results on the SQuAD for non-ensemble models.

mark. ELECTRA-Base also yields strong results, scoring substantially better than BERT-Base and XLNet-Base, and even surpassing BERT-Large according to most metrics. ELECTRA generally performs better at SQuAD 2.0 than 1.1. Perhaps replaced token detection, in which the model distinguishes real tokens from plausible fakes, is particularly transferable to the answerability classification of SQuAD 2.0, in which the model must distinguish answerable questions from fake unanswerable questions.

### 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 predicts 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 that are designed to be a set of "stepping stones" between BERT and ELECTRA.

- **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 5. 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

[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.

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

Table 5: Compute-efficiency experiments (see text for details).

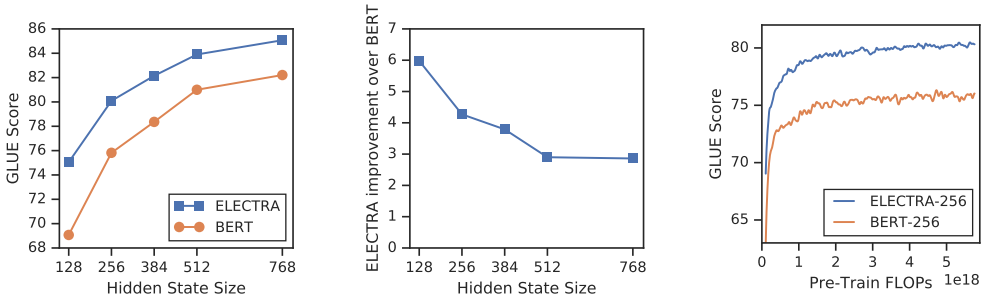


Figure 4: Left and Center: Comparison of BERT and ELECTRA for different model sizes. Right: A small ELECTRA model converges to higher downstream accuracy than BERT, showing the improvement comes from more than just faster training.

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, left). We find that the gains from ELECTRA grow larger as the models get smaller. The small models are trained fully to convergence (see Figure 4, right), showing that ELECTRA achieves higher downstream accuracy than BERT when fully trained. We speculate that ELECTRA is more parameter-efficient than BERT because it does not have to model the full distribution of possible tokens at each position, but we believe more analysis is needed to completely explain ELECTRA's parameter efficiency.

## 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., 2019a) and SpanBERT (Joshi et al., 2019) mask out contiguous sequences of token for improved span representations. This idea may be complementary to ELECTRA; we think it would be interesting to make ELECTRA's generator auto-regressive and add a "replaced span detection" task. 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. Recently, models such as TinyBERT (Jiao et al., 2019) and MobileBERT (Sun et al., 2019b) show that BERT can effectively be distilled down to a smaller model. In contrast, we focus more on pre-training speed rather than inference speed, so we train ELECTRA-Small from scratch.

**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. 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, 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). ELECTRA is particularly related to Noise-Contrastive Estimation (NCE) (Gutmann & Hyvärinen, 2010), which also trains a binary classifier to distinguish real and fake data points.

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.

## Acknowledgements

We thank Allen Nie, Prajit Ramachandran, audiences at the CIFAR LMB meeting and U. de Montréal, and the anonymous reviewers for their thoughtful comments and suggestions. We thank Matt Peters for answering our questions about ELMo, Alec Radford for answers about GPT, Naman Goyal and Myle Ott for answers about RoBERTa, Zihang Dai for answers about XLNet, Zhenzhong Lan for answers about ALBERT, and Danqi Chen and Mandar Joshi for answers about SpanBERT. Kevin is supported by a Google PhD Fellowship.

## References

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

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

Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, and Laurent Charlin. Language GANs falling short. *arXiv preprint arXiv:1811.02549*, 2018.

Jamie Callan, Mark Hoy, Changkuk Yoo, and Le Zhao. Clueweb09 data set, 2009. URL `https://lemurproject.org/clueweb09.php/`.

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

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 *NeurIPS*, 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.

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

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. In *NeurIPS*, 2019.

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

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

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 *NeurIPS*, 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.

Shankar Iyer, Nikhil Dandekar, and Kornl Csernai. First Quora dataset release: Question pairs, 2017. URL `https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs`.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

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.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

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. Technical report, Linguistic Data Consortium, Philadelphia, 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.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

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

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. MobileBERT: Task-agnostic compression of bert for resource limited devices, 2019b. URL `https://openreview.net/forum?id=SJxjVaNKwB`.

Guy Tevet, Gavriel Habib, Vered Shwartz, and Jonathan Berant. Evaluating text gans as language models. In *NAACL-HLT*, 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 *NeurIPS*, 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*, 2015.

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

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

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

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

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 ELECTRA models and BERT baselines. We mostly use the same hyperparameters as BERT. We set $\lambda$, 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 not improve scores (Yang et al., 2019; Liu et al., 2019). For our ELECTRA-Large model, we used a higher mask percent (25 instead of 15) because we noticed the generator was achieving high accuracy with 15% masking, resulting in very few replaced tokens. 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 $\lambda$ 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 6.

## B Fine-Tuning Details

For Large-sized models, we used the hyperparameters from Clark et al. (2019) for the most part. However, after noticing that RoBERTa (Liu et al., 2019) uses more training epochs (up to 10 rather than 3) we searched for the best number of train epochs out of [10, 3] for each task. For SQuAD, we decreased the number of train epochs to 2 to be consistent with BERT and RoBERTa. 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 found the small models benefit from a larger learning rate and searched for the best one out of [1e-4, 2e-4, 3e-4, 5e-3]. With the exception of number of train epochs, we used the same hyperparameters for all tasks. In contrast, previous research on GLUE such as BERT, XLNet, and RoBERTa separately searched for the best hyperparameters for each task. We expect our results would improve slightly if we performed the same sort of additional hyperparameter search. The full set of hyperparameters is listed in Table 7.

Following BERT, we do not show results on the WNLI GLUE task for the dev set results, as it is difficult to beat even the majority classifier using a standard fine-tuning-as-classifier approach. For the GLUE test set results, we apply the standard tricks used by many of the GLUE leaderboard submissions including RoBERTa (Liu et al., 2019), XLNet (Yang et al., 2019), and ALBERT (Lan et al., 2019). Specifically:

- For RTE and STS we use intermediate task training (Phang et al., 2018), starting from an ELECTRA checkpoint that has been fine-tuned on MNLI. For RTE, we found it helpful to combine this with a lower learning rate of 2e-5.

[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 | 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) | 1/4 | 1/3 | 1/4 |
| Mask percent | 15 | 15 | 25 |
| Learning Rate Decay | Linear | Linear | Linear |
| Warmup steps | 10000 | 10000 | 10000 |
| Learning Rate | 5e-4 | 2e-4 | 2e-4 |
| Adam $\epsilon$ | 1e-6 | 1e-6 | 1e-6 |
| Adam $\beta_1$ | 0.9 | 0.9 | 0.9 |
| Adam $\beta_2$ | 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 6: Pre-train hyperparameters. We also train an ELECTRA-Large model for 1.75M steps (other hyperparameters are identical).

| Hyperparameter | GLUE Value |
|---|---|
| Learning Rate | 3e-4 for Small, 1e-4 for Base, 5e-5 for Large |
| Adam $\epsilon$ | 1e-6 |
| Adam $\beta_1$ | 0.9 |
| Adam $\beta_2$ | 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, 2 for SQuAD, 3 for other tasks |

Table 7: Fine-tune hyperparameters

- For WNLI, we follow the trick described in Liu et al. (2019) where we extract candidate antecedents for the pronoun using rules and train a model to score the correct antecedent highly. However, different from Liu et al. (2019), the scoring function is not based on MLM probabilities. Instead, we fine-tune ELECTRA's discriminator so it assigns high scores to the tokens of the correct antecedent when the correct antecedent replaces the pronoun. For example, if the Winograd schema is "the trophy could not fit in the suitcase because it was too big," we train the discriminator so it gives a high score to "trophy" in "the trophy could not fit in the suitcase because the trophy was too big" but a low score to "suitcase" in "the trophy could not fit in the suitcase because the suitcase was too big."
- For each task we ensemble the best 10 of 30 models fine-tuned with different random seeds but initialized from the same pre-trained checkpoint.

While these tricks do improve scores, they make having clear scientific comparisons more difficult because they require extra work to implement, require lots of compute, and make results less apples-

to-apples because different papers implement the tricks differently. We therefore also report results for ELECTRA-1.75M with the only trick being dev-set model selection (best of 10 models), which is the setting BERT used to report results, in Table 8.

For our SQuAD 2.0 test set submission, we fine-tuned 20 models from the same pre-trained checkpoint and submitted the one with the best dev set score.

## C Details about GLUE

We provide further details about the GLUE benchmark tasks below

- **CoLA:** Corpus of Linguistic Acceptability (Warstadt et al., 2018). The task is to determine whether a given sentence is grammatical or not. The dataset contains 8.5k train examples from books and journal articles on linguistic theory.
- **SST:** Stanford Sentiment Treebank (Socher et al., 2013). The tasks is to determine if the sentence is positive or negative in sentiment. The dataset contains 67k train examples from movie reviews.
- **MRPC:** Microsoft Research Paraphrase Corpus (Dolan & Brockett, 2005). The task is to predict whether two sentences are semantically equivalent or not. The dataset contains 3.7k train examples from online news sources.
- **STS:** Semantic Textual Similarity (Cer et al., 2017). The tasks is to predict how semantically similar two sentences are on a 1-5 scale. The dataset contains 5.8k train examples drawn from new headlines, video and image captions, and natural language inference data.
- **QQP:** Quora Question Pairs (Iyer et al., 2017). The task is to determine whether a pair of questions are semantically equivalent. The dataset contains 364k train examples from the community question-answering website Quora.
- **MNLI:** Multi-genre Natural Language Inference (Williams et al., 2018). Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis, contradicts the hypothesis, or neither. The dataset contains 393k train examples drawn from ten different sources.
- **QNLI:** Question Natural Language Inference; constructed from SQuAD (Rajpurkar et al., 2016). The task is to predict whether a context sentence contains the answer to a question sentence. The dataset contains 108k train examples from Wikipedia.
- **RTE:** Recognizing Textual Entailment (Giampiccolo et al., 2007). Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis or not. The dataset contains 2.5k train examples from a series of annual textual entailment challenges.

## D Further results on GLUE

We report results for ELECTRA-Base and ELECTRA-Small on the GLUE test set in Table 8. Furthermore, we push the limits of base-sized and small-sized models by training them on the XLNet data instead of wikibooks and for much longer (4e6 train steps); these models are called ELECTRA-Base++ and ELECTRA-Small++ in the table. For ELECTRA-Small++ we also increased the sequence length to 512; otherwise the hyperparameters are the same as the ones listed in Table 6. Lastly, the table contains results for ELECTRA-1.75M without the tricks described in Appendix B. Consistent with dev-set results in the paper, ELECTRA-Base outperforms BERT-Large while ELECTRA-Small outperforms GPT in terms of average score. Unsurprisingly, the ++ models perform even better. The small model scores are even close to TinyBERT (Jiao et al., 2019) and MobileBERT (Sun et al., 2019b). These models learn from BERT-Base using sophisticated distillation procedures. Our ELECTRA models, on the other hand, are trained from scratch. Given the success of distilling BERT, we believe it would be possible to build even stronger small pre-trained models by distilling ELECTRA. ELECTRA appears to be particularly effective at CoLA. In CoLA the goal is to distinguish linguistically acceptable sentences from ungrammatical ones, which fairly closely matches ELECTRA's pre-training task of identifying fake tokens, perhaps explaining ELECTRA's strength at the task.

| Model | Train FLOPs | Params | CoLA | SST | MRPC | STS | QQP | MNLI | QNLI | RTE | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| TinyBERT | 6.4e19+ (45x+) | 14.5M | 51.1 | 93.1 | 82.6 | 83.7 | 89.1 | 84.6 | 90.4 | 70.0 | 80.6 |
| MobileBERT | 6.4e19+ (45x+) | 25.3M | 51.1 | 92.6 | 84.5 | 84.8 | 88.3 | 84.3 | 91.6 | 70.4 | 81.0 |
| GPT | 4.0e19 (29x) | 117M | 45.4 | 91.3 | 75.7 | 80.0 | 88.5 | 82.1 | 88.1 | 56.0 | 75.9 |
| BERT-Base | 6.4e19 (45x) | 110M | 52.1 | 93.5 | 84.8 | 85.8 | 89.2 | 84.6 | 90.5 | 66.4 | 80.9 |
| BERT-Large | 1.9e20 (135x) | 335M | 60.5 | 94.9 | 85.4 | 86.5 | 89.3 | 86.7 | 92.7 | 70.1 | 83.3 |
| SpanBERT | 7.1e20 (507x) | 335M | 64.3 | 94.8 | 87.9 | 89.9 | 89.5 | 87.7 | 94.3 | 79.0 | 85.9 |
| ELECTRA-Small | 1.4e18 (1x) | 14M | 54.6 | 89.1 | 83.7 | 80.3 | 88.0 | 79.7 | 87.7 | 60.8 | 78.0 |
| ELECTRA-Small++ | 3.3e19 (18x) | 14M | 55.6 | 91.1 | 84.9 | 84.6 | 88.0 | 81.6 | 88.3 | 63.6 | 79.7 |
| ELECTRA-Base | 6.4e19 (45x) | 110M | 59.7 | 93.4 | 86.7 | 87.7 | 89.1 | 85.8 | 92.7 | 73.1 | 83.5 |
| ELECTRA-Base++ | 3.3e20 (182x) | 110M | 64.6 | 96.0 | 88.1 | 90.2 | 89.5 | 88.5 | 93.1 | 75.2 | 85.7 |
| ELECTRA-1.75M | 3.1e21 (2200x) | 330M | **68.1** | **96.7** | **89.2** | **91.7** | **90.4** | **90.7** | **95.5** | **86.1** | **88.6** |

Table 8: Results for models on the GLUE test set. Only models with single-task finetuning (no ensembling, task-specific tricks, etc.) are shown.

## E COUNTING FLOPS

We chose to measure compute usage in terms of floating point operations (FLOPs) because it is a measure agnostic to the particular hardware, low-level optimizations, etc. However, it is worth noting that in some cases abstracting away hardware details is a drawback because hardware-centered optimizations can be key parts of a model's design, such as the speedup ALBERT (Lan et al., 2019) gets by tying weights and thus reducing communication overhead between TPU workers. We used TensorFlow's FLOP-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), but importantly, the forward/backward pass FLOPs really are 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.

## F 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. It is not possible to adversarially train the generator by back-propagating through the discriminator (e.g., as in a GAN trained on images) due to the discrete sampling from the generator, so we use reinforcement learning instead.

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

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

tokens $\{x_i : i \notin \boldsymbol{m}\}$, i.e., it does not depend on other generated tokens $\{\hat{x}_i : i \in \boldsymbol{m} \wedge i \neq t\}$. This isn't too bad of an assumption because a relatively small number of tokens are replaced, and it greatly simplifies credit assignment when using reinforcement learning. 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 we are assuming it does not depend on any replaced token. Therefore these tokens can be ignored when training $G$ to maximize $\mathcal{L}_{\text{Disc}}$. During training we seek to find

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

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

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{m},\hat{\boldsymbol{x}}} \left( \sum_{t \in \boldsymbol{m}} -\mathbb{1}(\hat{x}_t = x_t) \log D(\hat{x}|\boldsymbol{x}^{\text{masked}}) - \mathbb{1}(\hat{x}_t \neq x_t) \log(1 - D(\hat{x}|\boldsymbol{x}^{\text{masked}})) \right) \\
&= \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{m}} \sum_{t \in \boldsymbol{m}} \mathop{\mathbb{E}}_{\hat{x}_t \sim p_G} R(\hat{x}_t, \boldsymbol{x}) \\
&\text{where } R(\hat{x}_t, \boldsymbol{x}) = \begin{cases} -\log D(\hat{x}_t|\boldsymbol{x}^{\text{masked}}) & \text{if } \hat{x}_t = x_t \\ -\log(1 - D(\hat{x}_t|\boldsymbol{x}^{\text{masked}})) & \text{otherwise} \end{cases}
\end{aligned}$$

In short, the simplifying assumption allows us to decompose the loss over the individual generated tokens. We cannot directly find $\arg\max_{\theta_G}$ using gradient ascent because it is impossible to back-propagate 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 reward for the corresponding position. We approximate the expectations with a single sample and learn $\theta_G$ 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 58% 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).

## G 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 $x$ 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

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

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.

## H 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 added a prediction head to the model that predicted if the entire input was corrupted or not. Surprisingly, this slightly decreased scores on downstream tasks.

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