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

Anonymous authors
Paper under double-blind review

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 effective 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 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 while using less than 1/4 of the 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 (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 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 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).

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 et al., 2017).

1 Code and pre-trained weights will be release at [http://anonymized]
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 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 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.\footnote{It has 1/20th the parameters and requires 1/135th the pre-training compute of BERT-Large.} 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 $x = [x_1, ..., x_n]$ into a sequence of contextualized vector representations $h(x) = [h_1, ..., h_n]$. For a given position $t$, the generator outputs a probability for generating the token $x_t$ with a softmax layer:

$$p_G(x_t|\mathbf{x}) = \frac{\exp(e(x_t)^T h_G(\mathbf{x}))}{\sum_{x'} \exp(e(x')^T h_G(\mathbf{x}))}$$
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:

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

The generator is trained to perform masked language modeling (MLM). Given an input \( x = [x_1, x_2, ..., x_n] \), MLM first select a random set of positions (integers between 1 and \( n \)) to mask \( m = [m_1, ..., m_k] \). The tokens in the selected positions are replaced with a `[MASK]` token; we denote this as \( x^{\text{masked}} = \text{REPLACE}(x, m, [\text{MASK}]) \). 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 \( x^{\text{corrupt}} \) by replacing the masked-out tokens with generator samples and train the discriminator to predict which tokens in \( x^{\text{corrupt}} \) match the original input \( x \). Formally, model inputs are constructed according to

\[
m_i \sim \text{unif}\{1, n\} \text{ for } i = 1 \text{ to } k
\]

\[
x^{\text{masked}} = \text{REPLACE}(x, m, [\text{MASK}])
\]

and the loss functions are

\[
L_{\text{MLM}}(x, \theta_G) = \mathbb{E} \left( \sum_{i \in m} - \log p_G(x_i|x^{\text{masked}}) \right)
\]

\[
L_{\text{Disc}}(x, \theta_D) = \mathbb{E} \left( \sum_{t=1}^n \mathbb{I}(x_t^{\text{corrupt}} = x_t) \log D(x^{\text{corrupt}}, t) + \mathbb{I}(x_t^{\text{corrupt}} \neq x_t) \log (1 - D(x^{\text{corrupt}}, t)) \right)
\]

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

\[
\min_{\theta_G, \theta_D} \sum_{x \in \mathcal{X}} L_{\text{MLM}}(x, \theta_G) + \lambda L_{\text{Disc}}(x, \theta_D)
\]

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.

\[3\] Typically \( k = \lfloor 0.15n \rfloor \), i.e., 15% of the tokens are masked out.
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 D for more details on GLUE. Our evaluation metrics are Spearman correlation for STS, Matthews correlation for CoLA, and accuracy for the other GLUE tasks; we use “GLUE score” to refer to 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 un-answerable 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 (in fact, our pre-trained weights are drop-in replacements for BERT’s). See the appendix for further details and hyperparameter values. 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.

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 embeddings (both token embeddings and positional embeddings) of the generator and discriminator. In this case we use embeddings the size of the discriminator’s hidden states. 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 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

4We add linear layers to the generator to project the embeddings into generator-hidden-sized representations.
Interestingly, having a generator smaller than the discriminator improves results. 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 spend too 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 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:

1. Train only the generator with $L_{\text{MLM}}$ for $n$ steps.
2. Initialize the weights of the discriminator with the weights of the generator. Then train the discriminator with $L_{\text{Disc}}$ 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 discriminative objective, but does not end up outperforming joint training. Although still outperforming BERT, we found adversarial training to underperform maximum-likelihood training. Further analysis suggests the gap is caused by two 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. The second problem is that the adversarially trained generator produces a peaky 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 compare with DistilBERT, a smaller Transformer distilled from BERT-Base.\(^7\)

Results are shown in Table 1 (see Appendix D for a detailed breakdown of results on the test set). ELECTRA-Small performs remarkably well given its size, achieving a higher GLUE score than other methods using substantially more compute and parameters. ELECTRA-Small is trained mostly to convergence, with models trained for even less time (as little as 6 hours) still achieving reasonable performance. 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.

### 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 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-Large 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

---

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

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

7 https://medium.com/huggingface/distilbert-8cf3380435b5
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.

<table>
<thead>
<tr>
<th>Model</th>
<th>Train FLOPs</th>
<th>Params</th>
<th>CoLA</th>
<th>SST</th>
<th>MRPC</th>
<th>STS</th>
<th>QQP</th>
<th>MNLI</th>
<th>QNLI</th>
<th>RTE</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>1.9e20 (0.27x)</td>
<td>335M</td>
<td>60.6</td>
<td>93.2</td>
<td>88.0</td>
<td>90.0</td>
<td>91.3</td>
<td>86.6</td>
<td>92.3</td>
<td>70.4</td>
<td>84.0</td>
</tr>
<tr>
<td>XLNet</td>
<td>9.6e20 (3.1x)</td>
<td>360M</td>
<td>63.6</td>
<td>95.6</td>
<td>89.2</td>
<td>91.8</td>
<td>91.8</td>
<td>89.8</td>
<td>93.9</td>
<td>83.8</td>
<td>87.4</td>
</tr>
<tr>
<td>RoBERTa-100K</td>
<td>6.4e20 (0.90x)</td>
<td>356M</td>
<td>66.1</td>
<td>95.6</td>
<td>91.4</td>
<td>92.2</td>
<td>92.0</td>
<td>89.3</td>
<td>94.0</td>
<td>82.7</td>
<td>87.9</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>3.2e21 (4.5x)</td>
<td>356M</td>
<td>68.0</td>
<td>96.4</td>
<td>90.9</td>
<td>92.4</td>
<td>92.2</td>
<td>90.2</td>
<td>94.7</td>
<td>86.6</td>
<td>88.9</td>
</tr>
<tr>
<td>BERT (ours)</td>
<td>7.1e20 (1x)</td>
<td>335M</td>
<td>67.0</td>
<td>95.9</td>
<td>89.1</td>
<td>91.2</td>
<td>91.5</td>
<td>89.6</td>
<td>93.5</td>
<td>79.5</td>
<td>87.2</td>
</tr>
<tr>
<td>ELECTRA</td>
<td>7.1e20 (1x)</td>
<td>335M</td>
<td>69.3</td>
<td>96.0</td>
<td>90.6</td>
<td>92.1</td>
<td>92.4</td>
<td>90.5</td>
<td>94.5</td>
<td>86.8</td>
<td>89.0</td>
</tr>
</tbody>
</table>

Test set results for models with standard single-task finetuning (no ensembling, task-specific tricks, etc.)

<table>
<thead>
<tr>
<th>Model</th>
<th>Train FLOPs</th>
<th>Params</th>
<th>SQuAD 1.1</th>
<th>SQuAD 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EM</td>
<td>F1</td>
</tr>
<tr>
<td>BERT</td>
<td>1.9e20 (0.27x)</td>
<td>335M</td>
<td>80.8</td>
<td>88.5</td>
</tr>
<tr>
<td>SpanBERT</td>
<td>7.1e20 (1x)</td>
<td>335M</td>
<td>84.1</td>
<td>90.9</td>
</tr>
<tr>
<td>XLNet-Base</td>
<td>6.6e19 (0.09x)</td>
<td>117M</td>
<td>88.8</td>
<td>94.6</td>
</tr>
<tr>
<td>XLNet</td>
<td>6.6e19 (0.13x)</td>
<td>360M</td>
<td>81.3</td>
<td>–</td>
</tr>
<tr>
<td>RoBERTa-100k</td>
<td>6.4e20 (0.90x)</td>
<td>356M</td>
<td>85.0</td>
<td>94.5</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>3.2e21 (4.5x)</td>
<td>356M</td>
<td>–</td>
<td>94.0</td>
</tr>
<tr>
<td>BERT (ours)</td>
<td>7.1e20 (1x)</td>
<td>335M</td>
<td>88.9</td>
<td>94.6</td>
</tr>
<tr>
<td>ELECTRA-Base</td>
<td>6.4e19 (0.09x)</td>
<td>110M</td>
<td>88.0</td>
<td>93.7</td>
</tr>
<tr>
<td>ELECTRA</td>
<td>7.1e20 (1x)</td>
<td>335M</td>
<td>84.5</td>
<td>90.8</td>
</tr>
</tbody>
</table>

Table 3: Results on the SQuAD dev set for single models with no data augmentation or ensembling.

Results on SQuAD are shown in Table 3. Consistent, with the GLUE results, ELECTRA generally outperforms masked-language-modeling-based methods given the same compute resources. For example, ELECTRA-Large outperforms RoBERTa-100k and our BERT baseline, which use similar amounts of pre-training compute. ELECTRA-Large also roughly matches RoBERTa (performing 0.4 exact-match points better on SQuAD 2.0 and 0.2 points worse on SQuAD 1.1) despite using less than 1/4th of the compute. While SpanBERT [Joshi et al., 2019] performs slightly better at SQuAD 1.1, it performs significantly worse at 2.0 and its pre-training procedure is designed specifically to produce good span-level representations, which are important for question answering. We think it would be interesting future work to extend ELECTRA with a “replaced span detection” objective to improve its span representations. ELECTRA-Base also yields strong results, scoring substantially better than BERT-Base and XLNet-Base, and even surpassing than 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:
### Table 4: Compute-efficiency experiments (see text for details).

<table>
<thead>
<tr>
<th>Model</th>
<th>ELECTRA</th>
<th>All-Tokens MLM</th>
<th>Replace MLM</th>
<th>ELECTRA 15%</th>
<th>BERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLUE score</td>
<td>85.0</td>
<td>84.3</td>
<td>82.4</td>
<td>82.4</td>
<td>82.2</td>
</tr>
</tbody>
</table>

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 $L_{\text{Disc}}$ is over $i \in m$ instead of from 1 to $n$.

- **Replace MLM**: This objective is the same as masked language modeling except instead of replacing masked-out tokens with $[\text{MASK}]$, 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 $[\text{MASK}]$ 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 $[\text{MASK}]$ tokens and copy the input for other ones.

Results are shown in Table 4. 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 $[\text{MASK}]$ 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...

---

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 through 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 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


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.


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.


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.

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.
Table 5: Pre-train hyperparameters

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Small</th>
<th>Base</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of layers</td>
<td>12</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>Hidden Size</td>
<td>256</td>
<td>768</td>
<td>1024</td>
</tr>
<tr>
<td>FFN inner hidden size</td>
<td>1024</td>
<td>3072</td>
<td>4096</td>
</tr>
<tr>
<td>Attention heads</td>
<td>4</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>Attention head size</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Embedding Size</td>
<td>128</td>
<td>768</td>
<td>1024</td>
</tr>
<tr>
<td>Generator Size (multiplier for hidden-size, FFN-size, and num-attention-heads)</td>
<td>1/4</td>
<td>1/3</td>
<td>1/4</td>
</tr>
<tr>
<td>Learning Rate Decay</td>
<td>Linear</td>
<td>Linear</td>
<td>Linear</td>
</tr>
<tr>
<td>Warmup steps</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>5e-4</td>
<td>2e-4</td>
<td>2e-4</td>
</tr>
<tr>
<td>LAMB $\epsilon$</td>
<td>1e-6</td>
<td>1e-6</td>
<td>1e-6</td>
</tr>
<tr>
<td>LAMB $\beta_1$</td>
<td>0.9</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>LAMB $\beta_2$</td>
<td>0.999</td>
<td>0.999</td>
<td>0.999</td>
</tr>
<tr>
<td>Attention Dropout</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Weight Decay</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>Batch Size</td>
<td>128</td>
<td>256</td>
<td>2048</td>
</tr>
<tr>
<td>Train Steps (BERT/ELECTRA)</td>
<td>1.45M/1M</td>
<td>1M/766K</td>
<td>464K/400K</td>
</tr>
</tbody>
</table>

Table 6: Fine-tune hyperparameters

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>GLUE Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Rate</td>
<td>3e-4 for Small, 1e-4 for Base, 5e-5 for Large</td>
</tr>
<tr>
<td>Adam $\epsilon$</td>
<td>1e-6</td>
</tr>
<tr>
<td>Adam $\beta_1$</td>
<td>0.9</td>
</tr>
<tr>
<td>Adam $\beta_2$</td>
<td>0.999</td>
</tr>
<tr>
<td>Layerwise LR decay</td>
<td>0.8 for Base/Small, 0.9 for Large</td>
</tr>
<tr>
<td>Learning rate decay</td>
<td>Linear</td>
</tr>
<tr>
<td>Warmup fraction</td>
<td>0.1</td>
</tr>
<tr>
<td>Attention Dropout</td>
<td>0.1</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.1</td>
</tr>
<tr>
<td>Weight Decay</td>
<td>0</td>
</tr>
<tr>
<td>Batch Size</td>
<td>32</td>
</tr>
<tr>
<td>Train Epochs</td>
<td>10 for RTE and STS, 2 for SQuAD, 3 for other tasks</td>
</tr>
</tbody>
</table>

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 6.

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 \cite{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\textsuperscript{10} for our test set submission and compare with other methods that provide results in the same setting.

\section{Details on 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 sentence. The data dataset 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.

\section{Detailed Results on GLUE for ELECTRA-Small/Base}

We report results for ELECTRA-Base and ELECTRA-Small on the GLUE test set in Table\textsuperscript{7}. As in the dev-set results, ELECTRA-Base outperforms BERT-Large while ELECTRA-Small outperforms GPT in terms of average score. In fact, ELECTRA-Small even outperforms TinyBERT (Jiao et al., 2019), a small model distilled from BERT-base. TinyBERT uses a sophisticated distillation procedure for both pre-training and fine-tuning as well as data augmentation, while ELECTRA-Small uses standard training. As ablations on TinyBERT show it benefits largely from all three ideas, we expect a “TinyELECTRA” model also using distillation would perform even better. ELECTRA-Small is 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.

\section{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. We used TensorFlow’s

\textsuperscript{10}As with BERT, we select the best of 10 models to submit to the leaderboard on the dev set.
Table 7: Results for ELECTRA-Base and ELECTRA-Small on the GLUE test set. Only models with single-task finetuning (no ensembling, task-specific tricks, etc.) are shown.

<table>
<thead>
<tr>
<th>Model</th>
<th>Train FLOPs</th>
<th>Params</th>
<th>CoLA</th>
<th>SST</th>
<th>MRPC</th>
<th>STS</th>
<th>QQP</th>
<th>MNLI</th>
<th>QNLI</th>
<th>RTE</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyBERT</td>
<td>6.4e19+ (45x+)</td>
<td>14.5M</td>
<td>43.3</td>
<td>92.6</td>
<td>81.2</td>
<td>79.9</td>
<td>89.2</td>
<td>82.5</td>
<td>87.7</td>
<td>62.9</td>
<td>77.4</td>
</tr>
<tr>
<td>GPT</td>
<td>4.0e19 (29x)</td>
<td>117M</td>
<td>45.4</td>
<td>91.3</td>
<td>75.7</td>
<td>80.0</td>
<td>88.5</td>
<td>82.1</td>
<td>88.1</td>
<td>56.0</td>
<td>75.9</td>
</tr>
<tr>
<td>BERT-Base</td>
<td>6.4e19 (45x)</td>
<td>110M</td>
<td>52.1</td>
<td>93.5</td>
<td>84.8</td>
<td>85.8</td>
<td>89.2</td>
<td>84.6</td>
<td>90.5</td>
<td>66.4</td>
<td>80.9</td>
</tr>
<tr>
<td>BERT-Large</td>
<td>1.9e20 (135x)</td>
<td>335M</td>
<td>60.5</td>
<td>94.9</td>
<td>85.4</td>
<td>86.5</td>
<td>89.3</td>
<td>84.6</td>
<td>90.5</td>
<td>70.1</td>
<td>83.3</td>
</tr>
<tr>
<td>ELECTRA-Small</td>
<td>1.4e18 (1x)</td>
<td>14M</td>
<td>54.6</td>
<td>89.1</td>
<td>83.7</td>
<td>80.3</td>
<td>88.0</td>
<td>79.7</td>
<td>87.7</td>
<td>60.8</td>
<td>78.0</td>
</tr>
<tr>
<td>ELECTRA-Base</td>
<td>6.4e19 (45x)</td>
<td>110M</td>
<td>59.7</td>
<td>93.4</td>
<td>86.7</td>
<td>87.7</td>
<td>89.1</td>
<td>85.8</td>
<td>92.7</td>
<td>73.1</td>
<td>83.5</td>
</tr>
</tbody>
</table>

FLOPs-counting capabilities\[^1\] 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 \(2 \times m \times n\) FLOPs instead of \(m \times n\) 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 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 \(L_{\text{Disc}}\). 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-autoregressive: 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 \(D(\hat{x}_t | x_{\text{corrupt}}, t)\) depends only on the token \(x_t\) and the non-replaced tokens \(\{x_i : i \notin m\}\), i.e., it does not depend on other generated tokens \(\{\hat{x}_i : i \in m \land i \neq t\}\). 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 \(D(\hat{x}_t | x_{\text{masked}})\) for the discriminator predicting whether the generated token \(\hat{x}_t\) equals the original token \(x_t\) given the masked context \(x_{\text{masked}}\). A useful consequence of this assumption is that the discriminator score for non-replaced tokens (\(D(x_t | x_{\text{masked}})\) for \(t \notin m\)) 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 \(L_{\text{Disc}}\). During training we seek

\[^1\]See https://www.tensorflow.org/api_docs/python/tf/profiler
to find

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

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

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

$$= \mathbb{E}_{x, \mathbf{m}, \mathbf{\hat{x}} \sim P_G} R(\mathbf{\hat{x}}, x)$$

where 

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

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 back-propagate through discrete sampling of \(\mathbf{\hat{x}}\). Instead, we use policy gradient reinforcement learning (Williams, 1992). In particular, we use the REINFORCE gradient

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

Where \(b\) is a learned baseline implemented as \(b(x^{\text{masked}}, t) = \log \text{sigmoid}(w^T h_G (x^{\text{masked}}, t))\) where \(h_G (x^{\text{masked}})\) 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 on 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 section 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, and using any token masked-out, the discriminator loss can be written as:

$$\mathcal{L}_{\text{Disc}} = \sum_{x \in \text{vocabulary}} \left( (1 - p_{\text{mask}}) p_{\text{data}} (x|c) \log D(x, c) + p_{\text{mask}} p_{\text{data}} (x|c) p_{G} (x|c) \log D(x, c) + p_{\text{mask}} (1 - p_{\text{data}} (x|c)) p_{G} (x|c) \log (1 - D(x, c)) \right)$$

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
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. 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 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.

---

12 for ELECTRA-Base this means the upper-bound for accuracy is around 95%