Pretraining Deep Learning Models for Natural Language Understanding

Han Shao
Oberlin College
Oberlin, Ohio
hshao@oberlin.edu

Abstract—**Since the first bidirectional deep learning model for natural language understanding, BERT, emerged in 2018, researchers have started to study and use pretrained bidirectional autoencoding or autoregressive models to solve language problems. In this project, I conducted research to fully understand BERT and XLNet and applied their pretrained models to two language tasks: reading comprehension (RACE) and part-of-speech tagging (The Penn Treebank). After experimenting with those released models, I implemented my own version of ELECTRA, a pretrained text encoder as a discriminator instead of a generator to improve compute-efficiency, with BERT as its underlying architecture. To reduce the number of parameters, I replaced BERT with ALBERT in ELECTRA and named the new model, ALE (A Lite ELECTRA). I compared the performance of BERT, ELECTRA, and ALE on GLUE benchmark dev set after pretraining them with the same datasets for the same amount of training FLOPs.**

I. Introduction

Humans and computer systems communicate in fundamentally different approaches. Whereas we tell stories and relay information through a narrative, computers rely on objective data and commands. Bridging the gap is the root of natural language understanding. Traditionally, in the field of computational linguistics, researchers use statistical or rule-based modeling of natural language. The n-gram model is the simplest and most commonly used statistical model. It calculates the conditional probabilities of the next word for each n-word sequence. However, it can only construct the probabilities of the next word for a combination of n words that it has already seen during the training process. Meanwhile, n-gram models are lack of ability to capture long-range dependencies. Therefore, neural probabilistic language models were introduced by Bengio [1]. Specifically, each word is associated to a real-valued vector in R^m called word embedding. Moreover, the joint probability function of word sequences is expressed in terms of those word embeddings. Then, neural networks learn the word embeddings and the parameters in the probability function at the same time. By using this approach, seen words will increase the probability of similar words (similar words are expected to have similar word embeddings) to avoid n-gram model's shortages.

For years, the models constructing word embeddings are shallow neural networks and there is no need to use deep networks to create good embeddings [2]. Although word embeddings methods have great performance on the word-level, they lack the understanding of the interior connections among texts and are mostly task-specific. Hence, researchers proposed to build large-scale pretrained general models based on significant amount of text data to catch the trends of deep learning as the Computer Vision field does. In 2018, three pretrained models, ELMo [3], ULMFit [4], and GPT [5] were published by different AI research institutes. After small amount of training on specific datasets, they all accelerated the fine-tuning part and improved the performance. BERT [6] inherits the structure of GPT, and uses the same simple network as the underlying architecture, the Transformer [7], which is based solely on attention mechanisms, sparing recurrence and convolutions entirely.

In this project, I conducted research to fully understand the BERT model and applied its pretrained models (BERT_base and BERT_large) to two tasks: reading comprehension task (RACE [8]) and part-of-speech tagging task (The Penn Treebank [9]).

The average accuracy over 10 runs is 0.667 for RACE_middle by using BERT_base and 0.977 for the Penn Treebank by using BERT_large. While trying to boost BERT's performance, I also studied several improved versions of BERT including RoBERTa [10] and ALBERT [11]. Moreover, I noticed a autoregressive pretraining model named XLNet [12], while reading the related literature for natural language understanding. XLNet introduced a different method of language modeling called Permutation Language Modeling which is trained to predict one token given preceding context but conquer the shortness of unidirection by predicting tokens in random order. I applied XLNet_base on RACE_middle and XLNet_large on the Penn Treebank and compared XLNet's performance with BERT's.

After experimenting with pretrained models by downloading checkpoints from BERT and XLNet's websites, I started to implement my own version of ELECTRA [13], based on the descriptions in its paper. ELECTRA pretrains text encoders as discriminators rather than generators. It improves the efficiency of training process by learning from all tokens instead of the masked tokens (used by BERT) in the context. ELECTRA's architecture is the same as BERT's. To reduce the number of parameters, I proposed to replace BERT with ALBERT as ELECTRA's underlying structure and named the new model as ALE (A Lite version of ELECTRA). I compared the performance of BERT, ELECTRA, and ALE with the same pretraining datasets and the same amount of training FLOPs.

II. Models

In this project, I studied several versions of BERT and other two models using different language modelings, XLNet and ELECTRA.

A. BERT

BERT [6] stands for Bidirectional Encoder Representations from Transformers. Unlike traditional language models which process language in one direction, the bidirectional Transformer [7] allows BERT to process the sentence from both left to right and opposite, which improves the model's understanding of the above and following context where the word appears. In this work, we denote the number of layers as L, the hidden size as H, and the embedding size as E. BERT was pretrained by its group for two different sizes: BERT_base (L=12, E=H=768) and BERT_large (L=24, E=H=1024).

BERT's input is a concatenation of two segments. Each segment is consisted of word tokens. A [CLS] token is added to the beginning of the sequence, and two [SEP] tokens are used to separate segments. A general example of BERT's input would look like this: [CLS], t_1^A, ..., t_m^A, [SEP], t_1^B, ..., t_n^B, [SEP], [PAD], ..., [PAD]. [PAD] tokens will be added at the end when the sequence is shorter than the max sequence length. Moreover, each token will be mapped to a unique number according to the vocabulary file before being fed into the BERT models.

To achieve the bidirectional feature, the pretraining tasks of BERT are also different from former models'. Instead of predicting the next word given the preceding sequence, BERT utilizes a pretraining objective called Masked Language Model (MLM). MLM will mask 15% of word-piece tokens in each sequence randomly. BERT is trained to predict the masked tokens rather than reconstruct the entire input during pretraining.

Another pretraining task of BERT is next sentence prediction (NSP). The input of BERT is consisted of two segments. 50% of input's segment B are the actual following segment of segment A, and the rest of input's segment B are not. The NSP task is designed to allow BERT to learn the sentences relationships, though it is proved by RoBERTa group that NSP task is not very necessary.

The pretraining data BERT used is BooksCorpus (800M words) [14] and English Wikipedia (2500M words) [15].

B. RoBERTa

RoBERTa [10] stands for a Robustly Optimized BERT Pretraining Approach. Different from BERT, RoBERTa 1) uses more pretraining data totaling over 160GB (10x BERT pretraining data), 2) implements dynamic masking (to generate masked tokens every time feeding in the sequence) instead of static masking, 3) removes NSP loss, 4) is trained with large batches for longer time.

I studied this model but did not experiment with it since basically it has the same architecture as

BERT's but was trained with more data for longer time.

C. ALBERT

ALBERT [11] is a lite version of BERT. It reduces the number of parameters of BERT through factorized embedding parameterization and cross-layer parameter sharing.

In BERT, the embedding size E is tied with hidden layer size H. Therefore, E increases as H becomes large. The size of embedding table E $\times$ V (number of vocabularies) requires billions of parameters. ALBERT proposed a factorization of the embedding parameters. Instead of projecting the word token to a vector in size H directly, it first projects the word token to an embedding with size E, and then projects the embedding to a vector in size H. By using this decomposition, it reduces the space used by embedding table to E $\times$ V + E $\times$ H which is far smaller than H $\times$ V when V is large (over 30k).

Another approach of ALBERT to improve parameter efficiency is to share all parameters across layers. As a result, ALBERT_base only has 0.11x number of parameters of BERT_base.

In this project, I did not fine-tune ALBERT for downstream tasks, but used it to improve the parameter-efficiency of ELECTRA.

D. XLNet

fBERT has some limitations compared to an autoregressive (AR) model. First, BERT has an independent assumption since all masked tokens are reconstructed separately. On the contrary, an AR model uses the product rule without an independent assumption. Second, the input of BERT includes the symbol [MASK] which never appears in downstream tasks. In comparison, AR does not rely on input corruption. However, BERT also has its advantage compared to AR models. BERT can access to the whole context at the same time rather than only the seen first n tokens.

XLNet [12] uses Permutation Language Modeling (PLM) to take the advantages from both BERT and AR models and discard their shortages. By using PLM, XLNet was trained to predict the next word according to the previous sequence. However, different from traditional AR models, instead of predicting the words sequentially, it predicts the words in a random order.

XLNet also has two different sizes, XLNet_base and XLNet_large (sharing the same hyperparameters as BERT's). They were both pretrained on a larger dataset collections (160GB) compared to BERT's (16GB).

E. ELECTRA

The authors of ELECTRA [13] point out that MLM requires large amount of computations since models only learn from 15% of the input data by predicting the masked tokens. Therefore, they proposed a more compute-efficient pretraining task called replaced token detection. Instead of reconstructing the masked tokens, they used a small MLM as a generator to produce corrupted input by replacing each masked token with a word sampling from the probability distribution. Then, the corrupted input is fed into a discriminator which predicts whether each token was replaced by the generator or not. This pretraining task is more compute-efficient than MLM since the discriminator can learn from all tokens instead 15% tokens which are masked.

Both generator and discriminator in ELECTRA are built upon BERT. The generator is smaller than the discriminator and its size is controlled by a hyperparameter. This fraction will be multiplied to hidden size, number of attention heads, and intermediate size to enable the generator to be smaller than the discriminator while having the same number of layers as the discriminator does. A smaller generator is important to 1) reduce compute per training step, 2) not pose a too-challenging task for the discriminator preventing it from learning effectively.

III. Datasets

To test the performance of BERT and XLNet on reading comprehension and part-of-speech (POS) tagging tasks, I used RACE [8] and the Penn Treebank [9] datasets respectively. GLUE [16] benchmark dataset is used to evaluate the performance of ELECTRA, BERT, and ALE.

A. RACE

Large-scale ReAding Comprehension Dataset from Examination (RACE) was collected from English examinations in China, which are designed for middle school (RACE_middle) and high school (RACE_high) students . It is a large-scale reading comprehension dataset with more than 28,000 passages and nearly 100,000 questions [8].

RACE dataset has a higher level of difficulty compared to other reading comprehension datasets such as SQuAD1.0 [17] and SQuAD2.0 [18] because 1) all questions and candidate options are generated by human experts instead of extracting a segment from the original articles and generating wrong options randomly, 2) many questions require sentence-level reasoning, 3) articles' topics cover different domains and writing styles are also variant.

In this project, I trained and tested BERT_base on RACE_middle dataset. Table I lists the details of RACE_middle data. Each input example is stored in a single JSON file and is consisted of one article, mostly 4 questions, and a list of four candidate options for each question.

B. The Penn Treebank

The Penn Treebank [9] is a collection of sentences for which complete parse trees have been derived by expert human linguists. The parse trees indicate the grammatical structure of the sentences by marking the subject and predicate, prepositional phrases, etc. In creating the parse trees, the first step is to perform part-of-speech tagging (also done by human experts). Therefore, the dataset contains two forms of each sentence: one form with each word tagged with its POS and a second form which is the parse tree. The form that I used for the POS tagging task is the former one.

Each instance in the dataset is a sentence where each word is associated with a tag. The dataset contains 36 POS tags and 12 other tags (for punctuation and currency symbols). The difficulty of this task is that the same word in different sentences might have different tags. Therefore, it is a task that not only to find the connections between words and their tags but also to learn the word-wide relationships. The training set I used includes 45k sentences and the testing set includes 1k sentences. The averge sentence length is 24.4.

POS is believed to be an important part in language understanding. The meaning and the role of a word in a sentence is a small block which contributes to humans and computer systems' understanding of the whole sequence or even the whole context.

RACE_middle	**Train**	**Dev**
# articles	6409	368
# Questions	25421	1436
Passage Len	231.1	
Question Len	9.0	
Option Len	3.9	
Vocab Size	32811	

TABLE I
RACE_MIDDLE

Corpus	**|Train|**	**|Dev|**	**Task**	**Metrics**
Single-Sentence Tasks				
CoLA	8.5k	1k	acceptability	Mathews corr.
SST-2	67k	0.9k	sentiment	acc.
Similarity and Paraphrase Tasks				
MRPC	3.7k	0.4k	paraphrase	acc.
STS-B	5.7k	1.5k	sentence similarity	Pearson corr.
QQP	364k	40k	paraphrase	acc.
Inference Tasks				
MNLI	393k	10k	NLI	matched acc.
QNLI	105k	5.5k	QA/NLI	acc.
RTE	2.5k	0.3k	NLI	acc.

TABLE II
GLUE BENCHMARK TASKS

C. GLUE

The General Language Understanding Evaluation (GLUE) benchmark [16] is a collection of various natural language understanding tasks.

GLUE benchmark has three types of tasks: single-sentence tasks, similarity and paraphrase tasks, and inference tasks. Table II shows the detailed information of each task. The descriptions of tasks are listed in APPENDIX. Following BERT, I did not test the models on WNLI dev set since the train/dev split of WNLI is adversarial which results in weird behavior.

IV. EXPERIMENTS

A. BERT vs. XLNet

1) RACE: To fine-tune BERT on RACE, we need to convert the JSON files into the format of BERT's input. First, I read each JSON file as a single RaceExample which has properties:

- **id:** unique id for each example.
- **article:** the untokenized text of the article.

- **question:** the untokenized question.
- **options:** a list of four untokenized options.
- **label:** the correct answer (one of A, B, C, D).

Then, each RaceExample is processed to an InputFeature which has properties:

- **four_options:** a list of four Option objects. Each Option object has
  1) input_ids: a list of token ids.
  2) input_mask: if is padding (0) or actual tokens (1).
  3) segment_ids: segment A (0) or segment B (1).
- **label_id:** an integer in [0, 1, 2, 3] for four options.

A processed input example for BERT models: [CLS] tokenized article [SEP] tokenized question [SEP] tokenized option [SEP] [PAD]... [PAD]. The padding part ([PAD]s) will be added if the sequence length is smaller than the max sequence length.

After feeding the data into BERT, I extracted the output of all [CLS] tokens and fed those through a fully connected layer to compute the probabilities of four options for each question.

The BERT_base model I used has been pretrained by BERT group on BooksCorpus and English Wikipedia for 1M steps through MLM and NSP tasks. I fine-tuned BERT on RACE_middle dataset with sequence length 384, learning rate 5e-5, and number of epochs 3. The average accuracy over 10 runs is 0.667.

While fine-tuning XLNet on RACE, I used the XLNet group's implementation of applying XLNet to RACE. The data processing part is basically the same as BERT's except 1) if the question needs to fill in the blank, the blank is replaced by the options, 2) padding part is moved to the beginning of the sequence, 3) [PAD], [SEP], and [CLS] tokens have their own segment ids instead of 0 or 1. To keep the consistency, I trained and tested XLNet_base on RACE_middle dataset with learning rate 2e-5 for 2 epochs and achieved 0.692 average accuracy over 10 runs, 0.025 higher than BERT_base's average accuracy. It's difficult to compare the performance of XLNet and BERT on RACE dataset since XLNet was pretrained on 10x larger pretraining data. However, it shows that increasing the amount of training data is not an efficient method to improve downstream tasks' accuracy.

2) The Penn Treebank: The raw data of POS tagging task is a file consisted of thousands of sentences where the words in each sentence are labeled by POS tags. To fine-tune BERT on the Penn Treebank dataset, I added two more labels to the list of tags: ## and PAD. ## is used when a word is tokenized to multiple word-pieces such as importing $\Rightarrow$ import ##ing. ## is the label assigned to ##ing. Meanwhile, for the sequences whose length is smaller than max sequence length, label PAD will be assigned to [PAD] tokens.

From the training data file, each sentence will be converted to a PosExample which has properties:

- **id:** the unique id for each example.
- **sent:** a list of untokenized words in the sentence.
- **label:** a list of labels for all words.

Given each PosExample, I created an InputFeature which has properties:

- **input_ids:** [CLS] tokenized sent [PAD] ... [PAD].
- **input_mask:** 1 if the corresponding input_id has the original POS label, 0 otherwise (0 for ##token, [PAD] token, and [CLS] token.) It is used while calculating loss and accuracy.
- **segment_ids:** a list of 0s for one-sentence task.
- **label_li:** a list of labels. The original POS label for the first element of every tokenized word. ## for ##tokens. PAD for [PAD] and [CLS] tokens.

Different from RACE where only the output of [CLS] is used for predictions, all outputs from the last hidden layer of BERT are used for POS tagging predictions since it is a word-level problem rather than a sentence-level problem. Therefore, after feeding the processed data into BERT, the outputs of all

Model	**Dataset**	**LR**	**Num Epochs**	**acc.**
BERT_base	Race_middle	5e-5	3	0.667
XLNet_base	Race_middle	2e-5	2	0.692
XLNet_base	Treebank	5e-5	3	0.948
BERT_large	Treebank	5e-5	3	0.9773
XLNet_large	Treebank	5e-5	3	0.84

TABLE III
BERT vs. XLNet

Model	Train FLOPs	Params	MRPC	CoLA	MNLI	SST-2	QQP	QNLI	RTE	STS-B	Avg.
BERT_small	2.3e17	13.5M	0.838	0.317	0.785	0.908	0.877	0.862	0.610	0.862	0.757
ELECTRA_small	2.3e17	13.5M	0.824	0.509	0.770	0.877	0.888	0.847	0.592	0.840	0.768
2x time	4.6e17	13.5M	0.831	0.561	0.769	0.869	0.887	0.851	0.592	0.848	0.776
ALE_small	2.3e17	4.8M	0.738	0.305	0.745	0.834	0.878	0.824	0.567	0.810	0.713

TABLE IV
COMPARISON OF SMALL MODELS ON THE GLUE DEV SET

tokens are fed through a fully connected layer to calculate the probabilities of all tags for each word.

To get accurate loss and accuracy, I calculate the average by using input_mask (1 if the token has the original POS tag, 0 otherwise) as weights. I trained and tested BERT_large model on the Penn Treebank dataset with max sequence length 128, learning rate 5e-5 for 3 epochs. The average accuracy over 10 runs is 0.9773.

To improve the performance of BERT on POS tagging task, I also tried to replace the fully connected layer with a Conditional Random Field (CRF) [19] layer which will update a transition matrix during the training process. Each cell in the transition matrix stores the score of label transition from label X to label Y. After getting the outputs from BERT, the model will do the tag predictions based on logits and transition matrix while maximizing both the probability of label and the transition score from the last label to the current one. However, after applying this approach, the average accuracy over 10 runs is 0.9768. Using CRF layer did not improve the performance significantly. Its accuracy is even slightly lower than that while using a fully connected layer. It might be because our model has reached the ceiling performance on the Penn Treebank dataset for the POS tagging task.

The data processing of the Penn Treebank dataset for XLNet is exactly the same as BERT's, though some small modifications were made to fit XLNet input conventions such as the order of tokens and input mask format. To keep consistency, I trained and tested XLNet_large on Treebank dataset with learning rate 5e-5 for 3 epochs and achieved average accuracy 0.84 over 10 runs. Table III shows the results of both BERT and XLNet. XLNet_large's accuracy on POS tagging task is much lower than that of BERT_large's. Therefore, to test whether the bad results were caused by human errors, I trained and tested XLNet_base with the same experiment setting and achieved average accuracy 0.948 over 10 runs. My inference of XLNet_large's bad performance on the Penn Treebank dataset is that the model is large and has been trained on 160GB pretraining data before. Hence, it might be easy for it to become overfitting on a small downstream task dataset. Further investigation is required to verify this thought.

B. ELECTRA & ALE

ELECTRA and ALE are implemented by myself through TensorFlow library [20]. First, I prepared the pretraining datasets, BooksCorpus and English Wikipedia. BooksCorpus dataset is no longer publicly available, so I created my own BooksCorpus dataset by using a crawler to collect data from smashwords website [21]. English Wikipedia dataset is constructed by downloading the latest dump [15], extracting the text, and some cleanup steps. After reading all input files and concatenating all lines together, I converted the large corpus into sentence-per-line format which is required by BERT. The data processing is similar to what I did for BERT. Each sequence is tokenized, and [CLS], [SEP] and [PAD] tokens are added.

The basic architecture of ELECTRA is one generator and one discriminator. Both of them can be built upon BERT directly with some extra efforts:

1) **Implement Dynamic masking.** For each input sequence, I need to mask 15% tokens randomly every time the input is fed into the model.
2) **Make BERT into a MLM.** To build a generator, several layers need to be added to the end of BERT to make masked token predictions.
3) **Construct input for the discriminator**. The input for discriminator is generated by replacing masked tokens with word tokens sampling from the probability distribution provided by

the generator. I used temperature sampling [22] for this step. The temperature T was set to 0.8.

4) **Make BERT into a binary classification model.** To discriminate whether a token was replaced by the generator, a fully connected layer is added to the end of BERT.
5) **Build loss functions.** The loss functions of the generator and the discriminator was written according to the formulas in ELECTRA's paper.

ELECTRA has three different sizes, ELECTRA _small, ELECTRA_base, and ELECTRA_large. Due to the limitation of computing resources, I only pretrained ELECTRA_small and compared its performance on GLUE dev set with BERT_small's (trained by myself with the same pretraining data) after training for the same amount of FLOPs. To reduce the number of parameters, I came up with an idea to replace the underlying structure, BERT, with ALBERT in ELECTRA and called the new model ALE. The results (max of 10 runs) of ELECTRA_small, BERT_small, and ALE_small are shown in Table IV. The pretraining and fine-tuning hyperparameters are recorded in APPENDIX. From Table IV, we can notice that after training for the same amount of FLOPs on the same pretraining data, ELECTRA_small has better performance on GLUE than BERT_small does which implies learning from all tokens does increase the compute-efficiency during the pretraining process. Moreover, I also pretrained ELECTRA_small model for 2x longer time and its average result of GLUE improves which shows this model hasn't reached its ceiling performance and can still learn from the pretraining data. From table IV, we can also see that ALE_small's average result of GLUE is 0.713 which is fairly good if we consider the difference among numbers of parameters of three models. ALE_small has only 0.36x number of parameters compared to BERT_small or ELECTRA_small.

V. Conclusion

Pretrained deep learning models such as BERT and XLNet have gained the ability to solve the complex natural language understanding problems. By applying them to reading comprehension and POS tagging tasks, they achieved highest accuracy 0.692 for RACE and highest accuracy 0.977 for the Penn Treebank.

MLM conquers the weakness of the traditional AR models and enables BERT to learn from the context bidirectionally. Though BERT has noises ([MASK] tokens) in its input during pretraining process which will degrade the performance of downstream tasks and it is not that compute-efficient since it learns from only 15% of the tokens, XLNet and ELECTRA use different approaches to avoid these shortages. XLNet takes the advantages of both traditional AR models and BERT by using Permutation Language Modeling. ELECTRA proposed to train text encoders as discriminators rather than generators.

According to my implementation of ELECTRA and comparison between ELECTRA and BERT's performance on GLUE dataset, the improvement of compute-efficiency is verified. Moreover, after replacing BERT with ALBERT in ELECTRA, I save 64% space and keep a relatively good result.

From this project, I realized that though large pretrained models such as BERT and XLNet have achieved the state-of-the-art results on most NLU tasks, they both require a large amount of compute resources such as thousands of GPUs or TPUs and 160GB data for pretraining. ELECTRA shows another study direction for researchers that to optimize the machine learning process while using relatively small amount of compute. Pretraining ELECTRA_small with 16GB data requires only one GPU. By using ALE, it will save 64% parameters and still get a relatively good result.

In the future, I hope I could have chances to make developing and applying pretrained models more accessible to researchers and practitioners with less requirements to computing resources.

Acknowledgements

I thank Professor John Donaldson for his supports and encouragements during the whole project and Professor Adam Eck for his help on neural network implementations. I thank Kevin Clark for answers about ELECTRA and helping me to debug ELECTRA code. I thank Computer Science department at Oberlin College for providing me with an opportunity to spend an entire academic year conducting a research on NLU. I thank Google for the free access to a TPU on Google Colab and Google Cloud Platform.

REFERENCES

[1] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, no. null, p. 1137–1155, Mar. 2003.

[2] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *CoRR*, vol. abs/1708.02709, 2017. [Online]. Available: http://arxiv.org/abs/1708.02709

[3] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *CoRR*, vol. abs/1802.05365, 2018. [Online]. Available: http://arxiv.org/abs/1802.05365

[4] J. Howard and S. Ruder, "Fine-tuned language models for text classification," *CoRR*, vol. abs/1801.06146, 2018. [Online]. Available: http://arxiv.org/abs/1801.06146

[5] T. S. I. S. Alec Radford, Karthik Narasimhan, "Improving language understanding by generative pre-training," 2018.

[6] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[8] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy, "Race: Large-scale reading comprehension dataset from examinations," *arXiv preprint arXiv:1704.04683*, 2017.

[9] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Comput. Linguist.*, vol. 19, no. 2, p. 313–330, Jun. 1993.

[10] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019. [Online]. Available: http://arxiv.org/abs/1907.11692

[11] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=H1eA7AEtvS

[12] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *CoRR*, vol. abs/1906.08237, 2019. [Online]. Available: http://arxiv.org/abs/1906.08237

[13] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=r1xMH1BtvB

[14] Y. Zhu, R. Kiros, R. S. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," *CoRR*, vol. abs/1506.06724, 2015. [Online]. Available: http://arxiv.org/abs/1506.06724

[15] "Index of /enwiki/latest/." [Online]. Available: https://dumps.wikimedia.org/enwiki/latest

[16] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," *CoRR*, vol. abs/1804.07461, 2018. [Online]. Available: http://arxiv.org/abs/1804.07461

[17] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," *CoRR*, vol. abs/1606.05250, 2016. [Online]. Available: http://arxiv.org/abs/1606.05250

[18] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for squad," *CoRR*, vol. abs/1806.03822, 2018. [Online]. Available: http://arxiv.org/abs/1806.03822

[19] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, p. 282–289.

[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[21] "Smashwords – home." [Online]. Available: https://www.smashwords.com/

[22] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive Science*, vol. 9, pp. 147–169, 1985.

[23] W. B. Dolan and C. Brockett, "Automatically constructing a corpus of sentential paraphrases," in *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005. [Online]. Available: https://www.aclweb.org/anthology/I05-5002

[24] A. Warstadt, A. Singh, and S. R. Bowman, "Neural network acceptability judgments," *CoRR*, vol. abs/1805.12471, 2018. [Online]. Available: http://arxiv.org/abs/1805.12471

[25] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 1112–1122. [Online]. Available: https://www.aclweb.org/anthology/N18-1101

[26] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. [Online]. Available: https://www.aclweb.org/anthology/D13-1170

[27] S. Iyer, N. Dandekar, and K. Csernai, "First quora dataset release: Question pairs." [Online]. Available: https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs

[28] D. Giampiccolo, B. Magnini, I. Dagan, and B. Dolan, "The third PASCAL recognizing textual entailment challenge," in *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*. Prague: Association for

Computational Linguistics, Jun. 2007, pp. 1–9. [Online]. Available: https://www.aclweb.org/anthology/W07-1401

[29] D. M. Cer, M. T. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia, "Semeval-2017 task 1: Semantic textual similarity - multilingual and cross-lingual focused evaluation," *CoRR*, vol. abs/1708.00055, 2017. [Online]. Available: http://arxiv.org/abs/1708.00055

[30] Y. You, J. Li, J. Hseu, X. Song, J. Demmel, and C. Hsieh, "Reducing BERT pre-training time from 3 days to 76 minutes," *CoRR*, vol. abs/1904.00962, 2019. [Online]. Available: http://arxiv.org/abs/1904.00962

[31] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

APPENDIX

A. GLUE

- **MRPC:** Microsoft Research Paraphrase Corpus [23]. The input is consisted of two sentences. The task is to predict whether two sentences are semantically equivalent or not. The number of labels is two.
- **CoLA:** Corpus of Linguistic Acceptability [24]. This is an one-sentence task. It requires models to determine whether a given sentence is grammatical or not. This is a binary classification task.
- **MNLI:** Multi-genre Natural Language Inference [25]. The input is consisted of a premise sentence and a hypothesis sentence. The task is to predict whether the premise entails the hypothesis, contradicts the hypothesis, or neither. The number of labels is three.
- **SST-2:** Stanford Sentiment Treebank [26]. This is also an one-sentence task. It asks models to predict the sentiment of a given sentence is positive or negative.
- **QQP:** Quora Question Pairs [27]. The input is consisted of two questions. The task is to determine whether a pair of questions are semantically equivalent.
- **QNLI:** : Question Natural Language Inference; constructed from SQuAD [17]. Given a context sentence and a question sentence, the task is to predict whether the context sentence contains the answer to the question sentence.
- **RTE:** Recognizing Textual Entailment [28]. The input is consisted of a premise sentence and a hypothesis sentence. It needs models to predict whether the premise entails the hypothesis or not.
- **STS-B:** Semantic Textual Similarity [29]. Given two sentences, the task is to predict how semantically similar two sentences are on a range of 1-5. This is a regression problem.

B. ELECTRA & ALE

Hyperparams	ALE_small	ELECTRA_small	BERT_small
Num Params	4.8M	13.5M	13.5M
Max Seq Len	128	128	128
Embedding Size	128	128	128
Train Batch Size	1024	1024	1024
Train Steps	162k	125k	163k
Learning Rate	5e-4	5e-4	5e-4
Num Layers	12	12	12
Hidden Size	256	256	256
FNN Size	1024	1024	1024
Attention Heads	4	4	4
Att. Head Size	64	64	64
Generator Size	0.25	0.25	/
Mask Percent	15	15	15
LR Decay	Linear	Linear	Linear
Warmup Steps	10000	10000	10000
LAMB β_1	0.9	0.9	0.9
LAMB β_2	0.999	0.999	0.999
LAMB ϵ	1e-6	1e-6	1e-6
Dropout	0.1	0.1	0.1
Weight Decay	0.01	0.01	0.01

TABLE V
PRETRAINING HYPERPARAMETERS FOR SMALL MODELS

Task	Learning Rate	Train Batch Size	Num Epochs
MRPC	2e-4	64	3
CoLA	7e-5	32	3
MNLI	4e-4	256	3
SST-2	3e-4	256	3
QQP	5e-4	256	3
QNLI	2e-4	256	3
RTE	3e-4	32	10
STS-B	3e-5	8	10

TABLE VI
GLUE HYPERPARAMETERS

Table V and VI illustrate the details of hyperparameters of pretraining and GLUE train set for small models. The number of training steps are different since I want to train three models for the same amount of FLOPs. I used LAMB [30] as optimizer rather than ADAM [31] (ELECTRA's original optimizer) since the authors of ELECTRA mistook its optimizer as LAMB when the paper was under review.