How to Train BERT with an Academic Budget

Peter Izsak[§] **Moshe Berchansky**[§] **Omer Levy**[†]
[§] Intel Labs
{peter.izsak,moshe.berchansky}@intel.com
[†] Tel Aviv University
levyomer@cs.tau.ac.il

Abstract

While large language models à la BERT are used ubiquitously in NLP, pretraining them is considered a luxury that only a few well-funded industry labs can afford. How can one train such models with a more modest budget? We present a recipe for pretraining a masked language model in 24 hours, using only 8 low-range 12GB GPUs. We demonstrate that through a combination of software optimizations, design choices, and hyperparameter tuning, it is possible to produce models that are competitive with $\text{BERT}_{\text{BASE}}$ on GLUE tasks at a fraction of the original pretraining cost.

1 Introduction

Large language models, such as BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), and GPT-3 (Brown et al., 2020), have become the *de facto* models used in many NLP tasks and applications. However, their pretraining phase can be prohibitively expensive for startups and academic research groups, limiting the research and development of model pretraining to only a few well-funded industry labs. How can one train a large language model with commonly-available hardware in reasonable time?

We present a recipe for training a BERT-like masked language model in 24 hours, using only 8 Nvidia Titan-V GPUs (12GB each). Our approach combines multiple elements from recent developments: faster implementation (Rasley et al., 2020), faster convergence through overparameterization (Li et al., 2020b), single-sequence training (Joshi et al., 2020; Liu et al., 2019), and more. Moreover, we conduct an extensive hyperparmeter search tailored to our resource budget, and find that synchronizing the learning rate warmup and decay schedule with our 24 hour budget greatly improves model performance.

When evaluating on GLUE (Wang et al., 2018), our recipe produces models that are competitive with $\text{BERT}_{\text{BASE}}$ – a model that was trained on 16 TPUs for 4 days. Overall, our findings demonstrate that, with the right recipe, large language models can indeed be trained in an academic setting.[1]

2 Problem Setup

We investigate the task of pretraining a large language model under computational constraints. To simulate an academic computation budget, we limit the training time to 24 hours and the hardware to 8 low-range GPUs, specifically Nvidia Titan-V with 12GB memory each. When considering GB-hour as the currency, our setting is roughly equivalent to 3 days on a single 32GB Nvidia V100 GPU. Using the current prices of cloud services,[2] we estimate the dollar-cost of each training run at around $300 to $400.

Under these constraints, our goal is to pretrain a model that can benefit *classification* tasks, such as the ones in GLUE (Wang et al., 2018). Therefore, we follow the standard practice and focus on BERT-style transformer encoders trained on the masked language modeling objective (Devlin et al., 2019). We also retain the standard pretraining corpus of English Wikipedia and the Toronto BookCorpus (Zhu et al., 2015), containing 16GB of text, tokenized into subwords using BERT's uncased tokenizer.

3 Combining Efficient Training Methods

To speed up our training process, we combine a variety of recent techniques for optimizing a masked language model. While all of these techniques were reported by previous work, this is the first time, to the best of our knowledge, that they are combined and evaluated as a unified framework.

[1] Our code is available at: https://github.com/peteriz/academic-budget-bert

[2] Information gathered from https://cloud.google.com/compute/gpus-pricing and https://aws.amazon.com/ec2/instance-types/p3/ during April 2021.

3.1 Methods

Data Since our focus is primarily on sentence classification tasks, we limit the sequence length to 128 tokens for the entire pretraining process. Devlin et al. (2019) also apply this practice to 90% of the training steps, and extend the sequence to 512 tokens for the last 10%. This increases sample efficiency by reducing padding, and also allows us to fit a larger model into memory (see *Model*). In addition, we use single-sequence training without the next-sentence prediction objective, which was shown to benefit optimization by SpanBERT (Joshi et al., 2020) and RoBERTa (Liu et al., 2019). To reduce the amount of time spent on computing performance on the validation set, we held out only 0.5% of the data (80MB), and computed the validation loss once every 30 minutes.[3]

Model Recent work has found that larger models tend to achieve better performance than smaller models when trained for the same wall-clock time (Li et al., 2020b). We adopt these recommendations and train a $\text{BERT}_{\text{LARGE}}$ model: 24 layers, 1,024 dimensions, 16 heads, and 4,096 hidden dimensions in the feed-forward layer. Note that the purpose of applying the "train large" approach is *not* to compete with fully-trained extra-large models, but to train the best model we can – regardless of size – given the computational constraints in Section 2.

Optimizer We follow the optimization of RoBERTa (Liu et al., 2019) and use AdamW (Loshchilov and Hutter, 2017) with $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\varepsilon =$ 1e-6, weight decay of 0.01, dropout 0.1, and attention dropout 0.1. We experiment with various learning rates and warmups in Section 4.

Software We chose the DeepSpeed software package (Rasley et al., 2020), which includes various optimizations for training language models on multiple GPUs, such as I/O prefetching, mixed-precision training, and a transformer kernel for GPUs. We further modified parts of the implementation by replacing the masked language model prediction head with sparse token prediction, and using fused implementations for all linear-activation-bias operations and layer norms.

I/O To reduce the I/O bottleneck, we follow Devlin et al. (2019) and pre-mask 10 copies of the original corpus. This allows us to tensorize the entire training set and load it into RAM, rather than accessing the disk. While Liu et al. (2019) recommend dynamic masking, the benefits of applying it in our low-resource setting are marginal, and outweighed by the computational cost. Finally, we notice that the original data sharding implementation duplicates the corpus within each shard, resulting in highly homogeneous mini-batches; instead, we shuffle the entire dataset after the masking process to increase in-batch diversity.

[3] In the 5% of the training session (about 72 minutes) we compute the validation loss every 10 minutes.

	bsz	steps	samples	days
Google $\text{BERT}_{\text{BASE}}$	256	1000k	256M	5.85
Google $\text{BERT}_{\text{LARGE}}$	128†	2000k	256M	26.33
Our $\text{BERT}_{\text{LARGE}}$	128	2000k	256M	14.11
	256	1000k	256M	8.34
	4096	63k	256M	2.74
	8192	31k	256M	2.53
	16384	16k	256M	2.41

Table 1: A speed comparison between our optimized framework and the official implementation of BERT. Both implementations are run on our 8 GPU setting (12GB each), with the goal of covering 256 million training examples. We ignore the second phase of training in the official implementation, which extends the sequence length to 512 tokens and significantly prolongs the training time. †$\text{BERT}_{\text{LARGE}}$ does not fit in our GPUs with a batch sie of 256; we therefore measure according to the largest batch size we could fit (128), which requires twice as many steps to cover the same amount of training instances.

3.2 Combined Speedup

We compare our optimized framework to the official implementation code released by Devlin et al. (2019).[4] Table 1 shows that using the official code to train a base model could take almost *6 days* under the hardware assumptions described in Section 2, and a large model might require close to *a month* of non-stop computation. In contrast, our recipe significantly speeds up training, allowing one to train $\text{BERT}_{\text{LARGE}}$ with the original number of steps (1M) in a third of the time (8 days), or converge in 2-3 days by enlarging the batch size. While larger batch sizes do not guarantee convergence to models of equal quality, they are generally recommended in the literature (Ott et al., 2018; Liu et al., 2019), and present a more realistic starting point for our next phase (hyperparameter tuning) given our 24-hour constraint.

[4] https://github.com/tensorflow/models/tree/master/official/nlp/bert

4 Hyperparameter Search

Calibrating hyperparameters is key to increasing model performance in deep learning and NLP (Levy et al., 2015; Liu et al., 2019). We retune core optimization hyperparameters to fit our low-resource setting, rather than the massive computation frameworks for which they are currently tuned. Our hyperparameter search yields a substantial improvement in masked language modeling loss after 24 hours of training.

4.1 Hyperparameters

We tune the following hyperparameters:

Batch Size (bsz) The number of examples (sequences up to 128 tokens) in each mini-batch. Since we only have a limited amount of memory on each GPU, this can be inflated arbitrarily through gradient accumulation, at the cost of more time per update. We try batch sizes of 4,096, 8,192, and 16,384 examples. These batch sizes are of a similar order of magnitude to the ones used by RoBERTa (Liu et al., 2019).

Peak Learning Rate (lr) We use a linear learning rate scheduler, which starts at 0, warms up to the peak learning rate, and then decays back to 0. We try peak learning rates of 5e-4, 1e-3, and 2e-3.

Warmup Proportion (wu) We use a proportional system to determine the number of steps the learning rate warms up. We try 0%, 2%, 4%, and 6%. These proportions are significantly smaller than the proportion originally used by BERT (Devlin et al., 2019).

Total Days (days) The total number of days it would take the learning rate scheduler to decay back to 0, as measured on our hardware. This is equivalent to setting the maximal number of steps. Together with the warmup proportion, this hyperparameter determines where along the learning rate schedule the training process will cease. For a value of 1 day, the learning process will end exactly when the learning rate decays back to 0. We try setting the schedule according to a total number of 1, 3, and 9 days.

4.2 Methodology

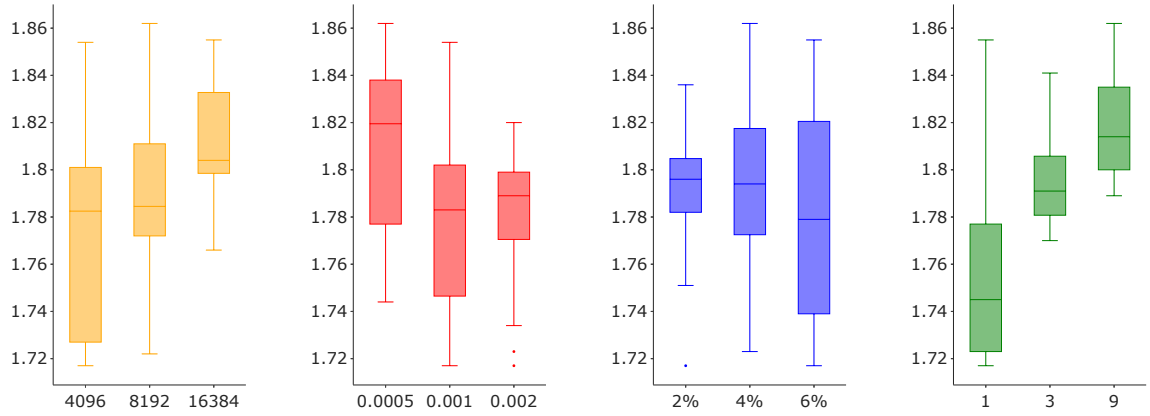
We optimize our model using masked language modeling (MLM) loss with each hyperparameter setting. While the grid of possible hyperparameter combinations spans 108 configurations, it is often easy to spot early on which runs are unlikely to reach top performance. After 3 hours, we prune configurations that did not reach a validation-set loss of 6.0 or less; this rule mainly removes diverging runs, such as configurations with 0% warmup. After 12 hours, we keep only the top 50% models with respect to validation loss, and resume their runs until they reach 24 hours. We report results based only on runs that completed 24 hours of training.

4.3 Results

We first analyze the effect of each hyperparameter by plotting the distribution of validation-set MLM loss per value (Figure 1). We observe a clear preference towards synchronizing the learning rate schedule with the actual amount of training time in the budget (1 day). As suggested by a previous work (Li et al., 2020a), annealing the learning rate to 0 at the end of the schedule may help the training process converge to a better model. We also find the smaller batch size to have an advantage over larger ones, along with moderate-high learning rates. We suspect that the smaller batch size works better for our resource budget due to the trade-off between number of samples and number of updates, for which a batch size of 4,096 seems to be a better fit. Finally, there appears to be a preference towards longer warmup proportions; however, a closer look at those cases reveals that when the number of total days is larger (3 or 9), it is better to use a smaller warmup proportion (2%), otherwise the warmup phase might take up a significantly larger portion of the *actual* training time.

Table 2 shows the best configurations by MLM loss. It is apparent that our calibrated models perform substantially better than models with BERT's default hyperparameters (which were tuned for 4 days on 16 TPUs). There is also relatively little variance in performance among the top models. We also observe, once again, that setting the number of training iterations to fit 1 day dominates the list.

We select the best-performing model (Search #1) for our downstream task evaluation, and name it **24hBERT**. For a final analysis, we compare the learning curve of our top model to that of prior configurations of BERT-style models. Figure 2 shows that 24hBERT converges significantly earlier and to a lower MLM loss.



(a) Batch Size (b) Learning Rate (c) Warmup Proportion (d) Total Days

Figure 1: Distribution of the validation-set MLM loss after 24 hours of training, when varying across different hyperparameters.

Configuration	loss	bsz	lr	wu	days
Search #1	1.717	4096	2e-3	6%	1
Search #2	1.717	4096	1e-3	2%	1
Search #3	1.720	4096	1e-3	6%	1
Search #4	1.722	8192	1e-3	6%	1
Search #5	1.723	4096	2e-3	4%	1
$\text{BERT}_{\text{BASE}}$	2.050	256	1e-4	11.1%	3
$\text{BERT}_{\text{LARGE}}$	2.318	256	1e-4	11.1%	8.3

Table 2: Hyperparameter configurations of the models with the lowest validation-set MLM loss recorded at the end of 24 hours. We compare different batch size *(bsz)*, peak learning rates *(lr)*, warmup proportions *(wu)* and total days for training *(days)* with BERT's original hyperparameters, using our combined optimized framework.

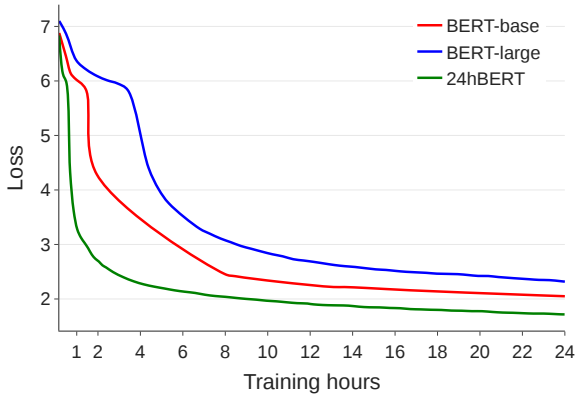


Figure 2: The validation-set MLM loss across training time, by different hyperparameter configurations.

5 Downstream Evaluation

We test the performance of our optimized, calibrated 24hBERT model on the GLUE benchmark (Wang et al., 2018).[5] For fine-tuning, we follow the practice of Liu et al. (2019) and run a grid search over multiple hyperparameters and seeds (see Appendix B for a complete list).

Table 3 shows the results on both validation and test sets of the various GLUE tasks. We observe that our 24hBERT model is able to perform on par with $\text{BERT}_{\text{BASE}}$ on 3 major datasets (MNLI, QNLI, SST-2) and even outperform it on 2 tasks (MRPC, CoLA), but reach lower performance than $\text{BERT}_{\text{BASE}}$ on 3 tasks (QQP, RTE, STS-B). The performance gap on RTE is especially large, and accounts for the difference in the average score.

[5]See Appendix C for a full description of the tasks.

We suspect that BERT's next-sentence prediction objective, which we omit (Section 3), is particularly helpful for bitext classification when only a few training examples (2.5k) are available. Overall, we find that our recipe produces a model that is largely competitive with $\text{BERT}_{\text{BASE}}$, but at a small fraction of its training cost.

6 Related Work

There are numerous works that aim to reduce the computation required for training BERT or similar models by using novel algorithmic approaches (Clark et al., 2020; Gong et al., 2019; Yang et al., 2020; Zhang and He, 2020; Kitaev et al., 2020; Wu et al., 2019; de Wynter and Perry, 2020). In contrast, our work explores the potential of optimizing a single BERT model. We assume our finding could be applied to other language mod-

	MNLI-m/mm	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	Avg.
#Examples	393k	105k	364k	2.5k	67k	3.7k	8.5k	7k	
$\text{BERT}_{\text{BASE}}$	84.6/84.0	90.6	72.0	69.1	92.8	86.1	55.1	84.3	79.8
$\text{BERT}_{\text{LARGE}}$	86.0/85.2	92.6	72.0	72.9	94.5	89.1	60.9	87.0	82.2
24hBERT	84.4/83.8	90.6	70.7	57.7	93.0	87.5	57.1	82.0	78.5

Table 3: Performance on GLUE test sets. All models are fine-tuned using the same hyperparameter space.

els trained on large corpora. As mentioned, many software optimization have already been shown in previous work (Shoeybi et al., 2019; Narayanan et al., 2021; Rasley et al., 2020; Ott et al., 2019). A recent work (Narang et al., 2021) exploring modifications to the Transformer architecture found that most modifications do not meaningfully improve performance of end-tasks. Despite their findings, we aim to combine methods that were proven to speed up the training process with little loss in accuracy.

7 Conclusions

We present a recipe for pretraining a masked language model in 24 hours using only 8 low-end GPUs. We show that by combining multiple efficient training methods presented in recent work and carefully calibrating the hyperparameters it is possible to pretrain a model that is competitive to $\text{BERT}_{\text{BASE}}$ on GLUE tasks. As with every recipe, our recommendations may need to be adapted to the hardware and time constraints at hand. We hope that our findings allow additional players to participate in language model research and development, and help democratize the art of pretraining.

References

Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, and Danilo Giampiccolo. 2006. The second pascal recognising textual entailment challenge. *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

Christopher Clark, Mark Yatskar, and Luke Zettlemoyer. 2020. Learning to model and ignore dataset bias with mixed capacity ensembles. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3031–3045, Online. Association for Computational Linguistics.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*, MLCW'05, page 177–190, Berlin, Heidelberg. Springer-Verlag.

Adrian de Wynter and D. Perry. 2020. Optimal subarchitecture extraction for bert. *ArXiv*, abs/2010.10499.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, RTE '07, page 1–9, USA. Association for Computational Linguistics.

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

Shankar Iyer, Nikhil Dandekar, and Kornl CsernaiZ. 2016. First quora dataset release: Question pairs.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *ArXiv*, abs/2001.04451.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

Mengtian Li, Ersin Yumer, and D. Ramanan. 2020a. Budgeted training: Rethinking deep neural network training under resource constraints. *ArXiv*, abs/1905.04753.

Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, K. Keutzer, D. Klein, and J. Gonzalez. 2020b. Train large, then compress: Rethinking model size for efficient training and inference of transformers. *ArXiv*, abs/2002.11794.

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

I. Loshchilov and F. Hutter. 2017. Fixing weight decay regularization in adam. *ArXiv*, abs/1711.05101.

Sharan Narang, Hyung Won Chung, Yi Tay, William Fedus, Thibault Fevry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam Shazeer, Zhenzhong Lan, Yanqi Zhou, Wei Li, Nan Ding, Jake Marcus, Adam Roberts, and Colin Raffel. 2021. Do transformer modifications transfer across implementations and applications?

D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, J. Bernauer, B. Catanzaro, Amar Phanishayee, and M. Zaharia. 2021. Efficient large-scale language model training on gpu clusters. *ArXiv*, abs/2104.04473.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 1–9, Brussels, Belgium. Association for Computational Linguistics.

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

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3505–3506, New York, NY, USA. Association for Computing Machinery.

M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *ArXiv*, abs/1909.08053.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

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

Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. *ArXiv*, abs/1901.10430.

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

Minjia Zhang and Yuxiong He. 2020. Accelerating training of transformer-based language models with progressive layer dropping.

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

Hyperparameter	Our Model
Number of Layers	24
Hidden size	1024
FFN inner hidden size	4096
Attention heads	16
Attention head size	64
Dropout	0.1
Attention Dropout	0.1
Learning Rate Decay	Linear
Weight Decay	0.01
Optimizer	AdamW
Adam ϵ	1e-6
Adam β_1	0.9
Adam β_2	0.98
Gradient Clipping	0.0
Batch Size	{4096, 8192, 16384}
Peak Learning Rate	{5e-4, 1e-3, 2e-3}
Warmup Proportion	{0%, 2%, 4%, 6%}
Max Steps	{24hr, 72hr, 216hr}

Table 4: Hyperparameters used for pretraining our models.

A Pretraining Configuration Hyperparameters

Table 4 presents the full set of hyperparameter configurations we examine in Section 4.

B Finetuning Hyperparameters

Finetuning hyperparameter used for the GLUE Benchmark tasks are presented in Table 5. We run each configuration using 5 random seeds and select the median of the best configuration.

C GLUE Tasks

The GLUE benchmark includes the following datasets, the descriptions of which were originally summarized by Wang et al. (2018):

MNLI: Multi-Genre Natural Language Inference is a large-scale, crowd-sourced entailment classification task (Williams et al., 2018). Given a pair of sentences, we wish to predict whether the second sentence is an entailment, contradiction, or neutral with respect to the first one.

QQP: Quora Question Pairs is a binary classification task, where the goal is to determine whether two questions asked on Quora are semantically equivalent or not (Iyer et al., 2016).

QNLI: Question Natural Language Inference is a version of the Stanford Question Answering Dataset (Rajpurkar et al., 2016). It has been converted into a binary classification task (Wang et al., 2018). The positive examples are (question, sentence) pairs, which contain the answer, and the negative examples are from the same paragraph, yet do not contain the answer.

SST-2: The Stanford Sentiment Treebank is a binary single-sentence classification task, consisting of sentences extracted from movie reviews. Their sentiment is based on human annotations (Socher et al., 2013).

CoLA: The Corpus of Linguistic Acceptability is a binary single-sentence classification task, where the goal is to predict whether an English sentence is linguistically "acceptable" or not (Warstadt et al., 2019).

STS-B: The Semantic Textual Similarity Benchmark is a collection of sentence pairs, drawn primarily from news headlines, woth additional sources as well (Cer et al., 2017). They were annotated with a score from 1 to 5, which denotes how similar the two sentences are, when semantic meaning is considered.

MRPC: Microsoft Research Paraphrase Corpus consists of sentence pairs automatically extracted from online news sources. The human annotations are for whether the sentences in the pair are semantically equivalent (Dolan and Brockett, 2005).

RTE: Recognizing Textual Entailment is a binary entailment task similar to MNLI, but with significantly less training data (Dagan et al., 2005; Bar-Haim et al., 2006; Giampiccolo et al., 2007).

Hyperparameter	**RTE, SST, MRPC, CoLA, STS, WNLI**	**MNLI, QQP, QNLI**
Learning Rate	{1e-5, 3e-5, 5e-5, 8e-5}	{5e-5, 8e-5}
Batch Size	{16, 32}	32
Weight Decay	0.1	0.1
Max Epochs	{3, 5, 10}	{3, 5}
Warmup Proportion	0.06	0.06

Table 5: The hyperparameter space used for finetuning our model on GLUE benchmark tasks.