How to Train BERT with an Academic Budget

Peter Izsak[♣] **Moshe Berchansky**[♣] **Omer Levy**[◇]
[♣]Intel Labs, Israel
[◇]Blavatnik School of Computer Science, Tel Aviv University
{peter.izsak,moshe.berchansky}@intel.com
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 a single low-end deep learning server. We demonstrate that through a combination of software optimizations, design choices, and hyperparameter tuning, it is possible to produce models that are competitive with BERT$_{\text{BASE}}$ on GLUE tasks at a fraction of the original pretraining cost.[1]

1 Introduction

Large language models, such as BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), and GPT3 (Brown et al., 2020), have become the *de facto* models used in many NLP tasks. 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 (MLM) in 24 hours in a limited computation environment. Our approach combines multiple elements from recent work: faster implementation (Rasley et al., 2020), faster convergence through over-parameterization (Li et al., 2020b), best practices for scaling language models (Kaplan et al., 2020), single-sequence training (Joshi et al., 2020; Liu et al., 2019), and more. Moreover, we conduct an extensive hyperparameter search tailored to our resource budget, and find that synchronizing learning rate warmup and decay schedules 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 BERT$_{\text{BASE}}$ – a model that was trained on 16 TPUs for 4 days. This recipe can also be applied to other corpora, as we demonstrate by training a French-language model on par with CamemBERT$_{\text{BASE}}$ (Martin et al., 2020) on the XNLI French benchmark (Conneau et al., 2018). Overall, our findings demonstrate that, with the right recipe and an understanding of the available computational resources, large language models can indeed be trained in an academic setting.

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 a single low-end deep learning server.[2] Using current cloud-compute prices, we estimate the dollar-cost of each training run at around \$50 to \$100.

Under these constraints, our goal is to pretrain a model that can benefit *classification* tasks, such as in GLUE (Wang et al., 2018). Therefore, we follow the standard practice and focus on BERT-style transformer encoders trained on the MLM objective (Devlin et al., 2019). We 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. To the best of our knowledge, this is the first time that such techniques are combined

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

[2]Specifically, we experiment with 8 Nvidia Titan-V GPUs with 12GB memory each. In terms of GB-hour, our setting is roughly equivalent to 1 day with 4 RTX 3090 GPUs or 2.4 days on a single 40GB A100 GPU.

and evaluated as a unified framework for training large models with limited computational resources.

3.1 Methods

Data Since our focus is mainly on sentence classification tasks, we limit sequences 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 (NSP) objective, which was shown to benefit optimization (Joshi et al., 2020; Liu et al., 2019). To maximize time spent on training, we hold out only 0.5% of the data and compute the validation-set loss every 30 minutes.

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; Kaplan et al., 2020). We adopt these recommendations and train a BERT$_{\text{LARGE}}$ model: 24 layers, 1,024 dimensions, 16 heads, 4,096 hidden dimensions in the feed-forward layer, with pre-layer normalization (Shoeybi et al., 2019). 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 (Section 2).

Optimizer We follow the optimization of RoBERTa (Liu et al., 2019) and use AdamW (Loshchilov and Hutter, 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\varepsilon = 1\text{e-}6$, weight decay of 0.01, dropout 0.1, and attention dropout 0.1. We experiment with various learning rates and warmups in Section 4. Preliminary experiments with other optimizers, such as LAMB (You et al., 2020), did not yield significantly different trends.

Software We base our implementation on the DeepSpeed software package (Rasley et al., 2020), which includes optimizations for training language models, such as data parallelization, and mixed-precision training. We further improve the implementation by replacing the MLM prediction head with sparse token prediction (Liu et al., 2019), and use fused implementations for all linear-activation-bias operations and layer norms, in particular the APEX LayerNorm operation.

	bsz	steps	samples	days
Google BERT$_{\text{BASE}}$	256	1000k	256M	5.85
Google BERT$_{\text{LARGE}}$	128†	2000k	256M	26.33
Our 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: Speed comparison between our optimized framework and the official implementation of BERT, while testing on the same hardware and controlling for the number of training examples covered (256M). †Largest batch size we could fit (128), requiring double the steps to cover the same amount of examples.

I/O To reduce the I/O bottleneck and minimize time wasted on non-training operations, we follow Devlin et al. (2019) and pre-mask 10 copies of the corpus. While Liu et al. (2019) recommends dynamic masking, the benefits of applying it in our low-resource setting are marginal, and outweighed by the computational cost. To ensure heterogeneous mini-batches, we shuffle the entire dataset after masking to remove intra-shard duplicates. Finally, we avoid disk I/O by sharding offline and loading the entire preprocessed dataset into RAM.

3.2 Combined Speedup

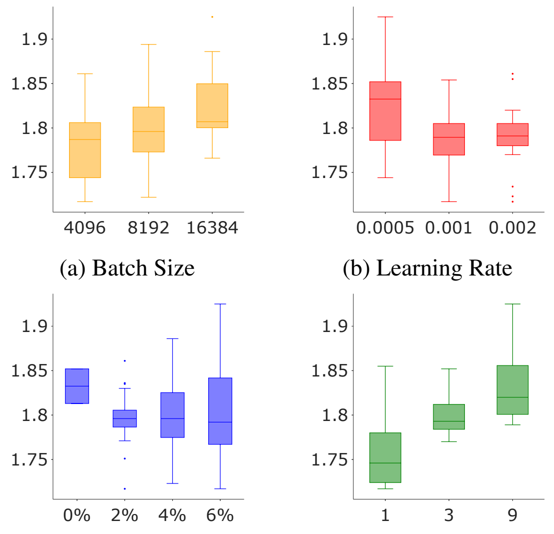
We compare our optimized framework to the official implementation of Devlin et al. (2019).[3] Table 1 shows that using the official code to train BERT$_{\text{BASE}}$ could take almost *6 days* under our hardware assumptions (Section 2), and a large model might require close to *a month* of non-stop computation. In contrast, our recipe significantly speeds up training, allowing one to train 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 (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.

We also conduct an ablation study of engineering improvements in our model. Table 2 shows that efficient implementation gains an additional 1.75 hours (out of 24) for training operations, which would have otherwise been wasted.

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

	Time Saved
– Sparse Output Prediction	-3.91%
– Fused Linear Layer	-4.39%
– APEX LayerNorm	-7.28%

Table 2: Speed up (in cumulative training time reduction) for each implementation improvement in our framework. Each line represents the original model without the measured feature, aggregated with preceding feature.



(a) Batch Size (b) Learning Rate

(c) Warmup Proportion (d) Total Days

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

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 re-tune 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 substantial improvements in MLM loss after 24 hours of training.

4.1 Hyperparameters

Batch Size (bsz) The number of examples (sequences up to 128 tokens) in each mini-batch. We try batch sizes of 4k, 8k, and 16k examples, which are of a similar order of magnitude to the ones used by Liu et al. (2019). Since our hardware has limited memory, we achieve these batch sizes via gradient accumulation. In terms of parameter updates, these batch sizes amount to approximately 23k, 12k, and 6k update steps in 24 hours, respectively.

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
BERT$_{\text{BASE}}$	2.050	256	1e-4	11.1%	3
BERT$_{\text{LARGE}}$	2.318	256	1e-4	11.1%	8.3

Table 3: Best hyperparameter configurations by MLM loss recorded after 24 hours of training.

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

Warmup Proportion (wu) We determine the number of warmup steps as a proportion of the total number of steps. Specifically, we try 0%, 2%, 4%, and 6%, which all reflect significantly fewer warmup steps than in BERT.

Total Days (days) The 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, it determines where along the learning rate schedule the training process stops. For a value of 1 day, the learning process will end when the learning rate decays back to 0. We try setting the schedule according to 1, 3, and 9 days.

4.2 Methodology

We optimize our model using MLM loss with each hyperparameter setting. Although there are 108 combinations in total, poor configurations are easy to identify early on. After 3 hours, we prune configurations that did not reach a validation-set loss of 6.0 or less; this rule removes diverging runs, such as configurations with 0% warmup. After 12 hours, we keep the top 50% of models with respect to the validation-set loss, and resume their runs until they reach 24 hours.

4.3 Results

We first analyze the effect of each hyperparameter by plotting the distribution of the validation-set 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), corroborating the results of Li et al. (2020a). We also find the smaller batch size to

#Examples	MNLI-m/mm 393k	QNLI 105k	QQP 364k	RTE 2.5k	SST-2 67k	MRPC 3.7k	CoLA 8.5k	STS-B 7k	Avg.
24hBERT	84.4/83.8	90.6	70.7	75.3	93.0	88.5	57.1	86.8	81.1
BERT$_{\text{BASE}}$	84.6/84.0	90.6	72.0	76.5	92.8	89.9	55.1	87.7	81.5
BERT$_{\text{LARGE}}$	86.0/85.2	92.6	72.0	78.3	94.5	89.9	60.9	87.5	83.0
RoBERTa$_{\text{BASE}}$	87.0/86.5	92.4	72.5	79.6	95.8	89.7	58.8	88.3	83.4

Table 4: Performance on GLUE test sets. Results for RTE, STS and MRPC are reported by first finetuning on the MNLI model instead of the baseline pretrained model.

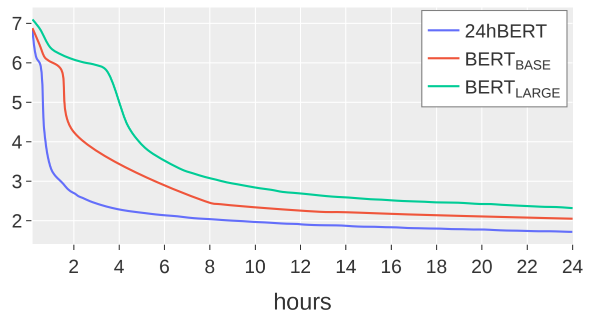


Figure 2: The validation-set loss of 24hBERT compared to the original BERT model configurations.

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 4096 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 larger portion of the *actual* training time.

Table 3 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 select the best model (Search #1), and name it **24hBERT**. Figure 2 compares 24hBERT with models using the default calibration, and shows that 24hBERT converges significantly faster.

5 Downstream Evaluation

We test the performance of our optimized, calibrated 24hBERT model on the GLUE benchmark (Wang et al., 2018).[4] For finetuning, we follow the practice of Liu et al. (2019), and run a grid search over multiple hyperparameters and seeds (see Appendix B), and also use mid-training (Phang et al., 2018) on MNLI for RTE, MRPC and STS-B.

Table 4 shows the results on GLUE's test sets. Our 24hBERT model performs on par with BERT$_{\text{BASE}}$ on 3 major tasks (MNLI, QNLI, SST-2) and even outperforms it on CoLA. However, 24hBERT reaches slightly lower results on 4 tasks (QQP, RTE, MRPC, STS-B). Overall, this amounts to a small difference on the average score (0.4%), showing that our recipe can indeed produce a model that is largely competitive with BERT$_{\text{BASE}}$, but at a small fraction of its training cost.

6 Generalizing to New Corpora

Our recipe was calibrated using a particular corpus (English Wikipedia and books), but does it generalize to other corpora as well? We follow CamemBERT (Martin et al., 2020) and train a masked language model on French Wikipedia, using exactly the same dataset. We then finetune our French 24hBERT on the XNLI French dataset (Conneau et al., 2018), reaching 78.5% accuracy, compared to 79.1% of CamemBERT$_{\text{BASE}}$. This result demonstrates that our recipe can indeed be ported to other corpora as-is, without retuning hyperparameters.

7 Discussion

Comparison with ELECTRA While Clark et al. (2020) show impressive pretraining speedups with ELECTRA, we argue that having a generative model (MLM or LM) is important nowadays, given the recent rise of few-shot learning and prompting approaches (Schick and Schütze, 2021). To emphasize this point, we run 24hBERT on the SST-2 (Socher et al., 2013) task both with and without prompts in the few-shot setting. Figure 3 shows that there is a significant advantage in the ability to prompt the model, which is perhaps not trivial for non-generative ELECTRA-style models.

[4]See Appendix D for a full description of tasks.

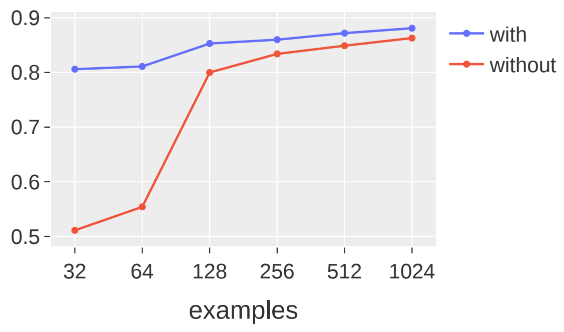


Figure 3: The performance of 24hBERT on the SST-2 task in few-shot settings (5 seeds for each #examples, with 25% of the examples used for validation), with and without prompts.

FLOPs as a Measure of Efficiency While measuring floating point operations is commonly used to compare efficiency in a hardware-agnostic manner, it is *not* an accurate tool for comparing the actual time (and therefore budget) associated with training a model. Specifically, measuring FLOPs ignores the fact that many operations run in parallel (e.g. via batching), and are thus much less costly in practice (Li et al., 2020b).

Limitations Our investigation is limited to classification tasks. While it is true that it is not fully comparable with BERT$_{\text{BASE}}$ in that using short sequences does not allow for reading comprehension tasks (without resorting to sliding windows), it might be possible to continue training the model for a few more hours with sequences longer than 128 tokens, as done by Devlin et al. (2019). We leave such experiments for future work.

8 Conclusions

We present a recipe for pretraining a masked language model in 24 hours using a low-end deep learning server. 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 BERT$_{\text{BASE}}$ on GLUE tasks. In contrast to other works in this area, which often focus a single method for improving efficiency, our recipe consists of many different components that together amount to very large speedups:

- Short sequences (Devlin et al., 2019)
- Single-sequence training (Joshi et al., 2020)
- Training larger models (Li et al., 2020b)
- DeepSpeed (Rasley et al., 2020)
- Sparse token prediction (Liu et al., 2019)
- Fused implementations
- Avoiding disk I/O
- Large batch sizes (Liu et al., 2019)
- Large learning rates (Liu et al., 2019)
- Short warmup
- Synchronizing schedule with time budget (Li et al., 2020a)

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.

Kevin Clark, Minh-Thang Luong, Quoc Le, and Christopher D. Manning. 2020. Pre-training transformers as energy-based cloze models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 285–294, Online. Association for Computational Linguistics.

Alexis Conneau, Guillaume Lample, Ruty Rinott, Adina Williams, Samuel R. Bowman, Holger Schwenk, and Veselin Stoyanov. 2018. Xnli: Evaluating cross-lingual sentence representations.

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.

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.

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.

J. Kaplan, Sam McCandlish, T. Henighan, Tom B. Brown, Benjamin Chess, R. Child, Scott Gray, Alec Radford, Jeff Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *ArXiv*, abs/2001.08361.

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 Deva Ramanan. 2020a. Budgeted training: Rethinking deep neural network training under resource constraints. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez. 2020b. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5958–5968. PMLR.

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. 2019. Decoupled weight decay regularization. In *ICLR*.

Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric de la Clergerie, Djamé Seddah, and Benoît Sagot. 2020. CamemBERT: a tasty French language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7203–7219, Online. 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.

Jason Phang, Thibault Févry, and Samuel R. Bowman. 2018. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088v2*.

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.

Timo Schick and Hinrich Schütze. 2021. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269, Online. Association for Computational Linguistics.

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.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2020. Large batch optimization for deep learning: Training bert in 76 minutes.

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.

A Pretraining Hyperparameters

Table 5 presents the full set of hyperparameter configurations we examine in Section 4.

B Finetuning Hyperparameters

Finetuning hyperparameters used for the GLUE benchmark tasks are presented in Table 7. We run each configuration using 5 random seeds and select the median of the best configuration.

C Performance Comparison

Table 6 includes time comparison of our 24 hour training setup when using more recent hardware backends.

D Downstream Tasks

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, with 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.

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 5: Hyperparameters used for pretraining our models.

	GPUs	**Days**	**BSZ/GPU**	**ACC**
Titan-V 12GB	8	1.00	32	16
RTX 3090 24GB	1	5.84	112	37
	4	1.55	112	9
A100 40GB	1	2.75	200	20
	4	0.74	200	5

Table 6: Number of days, batch size per GPU (BSZ/GPU), and number of gradient accumulations (ACC) to train a model using our recipe (24hBERT) with more recent GPUs.

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).

XNLI French: Cross-lingual Natural Language Inference French, an entailment classification task (Conneau et al., 2018) similar to MNLI, with that the premise and hypothesis in each example are in the French language.

Hyperparameter	**RTE, SST-2, MRPC, CoLA, STS-B, WNLI**	**MNLI, QQP, QNLI, XNLI French**
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 7: The hyperparameter space used for finetuning our model on GLUE benchmark tasks, and the XNLI French task.