Summary is provided at Pg 13. It is recommended to go through the summary first before reading the whole paper

CRAMMING: TRAINING A LANGUAGE MODEL ON A SINGLE GPU IN ONE DAY

Jonas Geiping
University of Maryland, College Park
jgeiping@umd.edu

Tom Goldstein
University of Maryland, College Park
tomg@umd.edu

ABSTRACT

Recent trends in language modeling have focused on increasing performance through scaling, and have resulted in an environment where training language models is out of reach for most researchers and practitioners. While most in the community are asking how to push the limits of extreme computation, we ask the opposite question: How far can we get with a single GPU in just one day?

Focus of the paper

We investigate the downstream performance achievable with a transformer-based language model trained completely from scratch with masked language modeling for a *single* day on a *single consumer* GPU. Aside from re-analyzing nearly all components of the pretraining pipeline for this scenario and providing a modified pipeline with performance close to BERT, we investigate why scaling down is hard, and which modifications actually improve performance in this scenario. We provide evidence that even in this constrained setting, performance closely follows scaling laws observed in large-compute settings. Through the lens of scaling laws, we categorize a range of recent improvements to training and architecture and discuss their merit and practical applicability (or lack thereof) for the limited compute setting.

1 SCALING UP AND SCALING DOWN

Large-scale training of machine learning models with transformer architectures has lead to groundbreaking improvements in many sub-fields of natural language processing including language understanding and natural language generation (Vaswani et al., 2017; Dosovitskiy et al., 2021; Radford et al., 2019). The nowadays accepted (but historically surprising) key behavior of these systems is that they reliably *scale* – they continuously improve in performance when the number of model parameters and amount of data grow. These increases in performance are well-described by various power laws as studied by Kaplan et al. (2020). This sets up a dominant paradigm in which scaling is the key to performance improvement (Sutton, 2019).

Reasons why scaling LLMs gets a lot of attention these days

The power of scale has set off a race to produce extremely large models, which in turn has created an environment where few researchers or practitioners feel that they are capable of training a language model. The original BERT model Devlin et al. (2019), which became a cornerstone transformer for many practical applications in natural language understanding, already required a significant amount of computation to train. Yet, the reproduction and improvements in Liu et al. (2019) further increased its performance by cranking up the level of computation by orders of magnitude. As these pre-trained checkpoints became popular for a range of downstream applications (Wolf et al., 2020), the competition for the largest language model became a focal point for industrial labs. This led to training runs that improved the performance of pretrained language models at the expense of computation at the zettaFLOP scale (Raffel et al., 2020; Yang et al., 2020; Zaheer et al., 2021) and later at the extremely large yottaFLOP scale (Brown et al., 2020; Black et al., 2022; Chowdhery et al., 2022; Rae et al., 2022).

Our goal is to turn this trend on its head and investigate how to best *scale down* language model training and what trade-offs emerge when doing so: *What downstream performance can be achieved by a modest researcher when training from scratch with a single GPU for a single day?* The ability to train a language model to the performance level of BERT with such modest resources has several interesting implications. For one, if scaled-down model pretraining is a viable analogue of large-compute pretraining, then this opens up a host of further academic investigations that are currently hard to realize for large-scale models. For example, research questions about the differences between existing and new pre-training tasks, tracing model predictions to data points (Ilyas et al., 2022), security questions such as membership inference (Carlini et al., 2022) and data poisoning (Geiping et al., 2021), and a wide range of empirical investigations into topics such as stability or generalization that arise during training (Nagarajan & Kolter, 2019; Jiang et al., 2019). At the same time, we can imagine situations in which legal requirements make it unclear whether models trained on public data with uncertain origin are permissible, and where a practitioner is interested in retraining their language models using a specialized or trustworthy data source (Wilka et al., 2017; Gold & Latonero, 2017).

Reasons why scaling down is necessary and more interesting aspect than simply scaling up

In addition, we are motivated to benchmark the overall *conceptual* progress of research in this area over the last years, beyond simply turning the scaling knob. The goal of achieving BERT-like performance with modest training resources would have seemed unthinkable in 2018, and yet with modern advances and transformer training techniques this may now be possible.

To answer these questions, we consider a challenge we call "Cramming" – learning a whole language model the day before the test. Our studies begin by investigating many facets of the training pipeline to see which modifications actually improve performance in the scaled-down scenario. We provide evidence that even in this constrained setting, performance closely follows scaling laws observed in large-compute settings. An unsurprising consequence of these laws is that scaling down is hard; while smaller model architectures enable speeding up gradient computations, overall rates of model improvement over time remain nearly constant. Nonetheless, we can find changes to the training recipe that exploit scaling laws to yield improvements by improving the effective rate of gradient computations without compromising model size. In the end, we are able to train models that achieve respectable performance – often close to and sometimes exceeding BERT on GLUE tasks – on a shoestring budget[1].

2 Tying our hands behind our back: A setup with limited compute

Before we start this investigation, we want to outline the extent of limitations we are interested in. The rules for cramming are as follows:

A very nice minimal setup indeed! Good to see this coming from academia side.

- A transformer-based language model of arbitrary size is trained with masked-language modeling, completely from scratch.
- Existing pretrained models cannot be included in any part of the pipeline.
- Any raw text (excluding downstream data) can be included for training. This means that one can achieve speedups by making judicious choices about how and when to sample data, provided the sampling mechanism does not require a pre-trained model.
- The downloading and pre-processing of raw data is exempted from the total compute budget. Pre-processing may include CPU-based tokenizer construction, tokenization, and filtering, but cannot include representation learning (e.g. pre-training a word embedding is not allowed, unless it is counted towards the final runtime).
- Training proceeds on a single GPU for 24 hours.
- Downstream performance is evaluated on GLUE (Wang et al., 2018). Downstream finetuning on GLUE is limited to brief training with only the training data of the downstream task (we consider 5 epochs or less) and needs to work with hyperparameters set globally for all GLUE tasks. Downstream finetuning is excluded from the total compute budget.

In our implementation, we analyze both a setup with a classical `rtx2080ti` GPU (released September 2018) and separate setups with a more modern `rtxa4000` or `rtxa6000` GPU (released October 2020). We pair each unit with 4 CPU cores and 32GB of RAM.

Why we need to have a minimal setup?

Why these limitations? We are principally interested in re-investigating the original BERT setup of Devlin et al. (2019) with limited compute. The optimal architecture of the transformer is not fixed,

[1] We provide code to replicate all experiments at https://github.com/JonasGeiping/cramming.

Only things to notice in this table

Group	Target	Accelerator	Time Limit	Total exaFLOP
(Devlin et al., 2019)	BERT	16 TPU	4 days	680
(Dettmers, 2018)	BERT	8 V100	11 days	950
(Narasimhan, 2019)	BERT-large	1472 V100	47 min	519
(Raffel et al., 2020)	T5-base	16 TPUv3	1 day	170
(Iandola et al., 2020)	squeezeBERT	8 Titan RTX	4 days	361
(Narang et al., 2021)	T5 variations	16 TPUv3	1.75 days	298
(Tay et al., 2021)	T5-small-L16	16 TPUv3	11.2 hours	82
(Izsak et al., 2021)	BERT variation	8 V100	1 day	86
(Liu et al., 2019)	roBERTa-base	1024 V100	1.25 day	13 824
(Chowdhery et al., 2022)	PaLM	6144 TPUv4	50 days	7 299 072
Our Setup 1	BERT variation	1 rtx2080ti	1 day	5
Our Setup 2	BERT variation	1 rtxa4000	1 day	8
Our Setup 3	BERT variation	1 rtxa6000	1 day	13

Table 1: Maximal Throughput available for select training runs of large language models. FLOP Counts for BERT reproductions and related models. Large-scale LMs included only for reference.

as the optimal size and shape depends on scaling laws (Kaplan et al., 2020). The limitations on usage of existing models rule out distillation from an existing model (Turc et al., 2019; Jiao et al., 2020; Sun et al., 2020; Wang et al., 2020b; Kaliamoorthi et al., 2021) and data filtering based on existing large models (Golchin et al., 2022), both of which ultimately answer questions about compression and transfer of already processed information. Further, we do not want to limit data to the original dataset used to train BERT, wanting to allow for possible improvements through better data curation and quality. The rtx2080ti GPU is a natural candidate for this experiment, given that it was released before Devlin et al. (2019), but the more recent rtxa4000 is also interesting, as a more recent consumer-grade workstation variant. Finally we also test the rtxa6000, being arguably the upper limit of a single-user workstation. At the finetuning stage we want to mimic the original BERT finetuning and evaluation setup, but provide additional limits to prevent gains based on tuning of only the downstream procedure, for example via computationally extensive downstream training (Bahri et al., 2021a), use of multiple downstream datasets (for example continued pretraining with MNLI before finetuning other tasks (Izsak et al., 2021)), and extended hyperparameter optimization for each GLUE task (Devlin et al., 2019; Liu et al., 2019; Lan et al., 2019).

3 Related Work on Efficient Transformers

How long does it take to train BERT? In general, this question is hard to answer, due to wildly varying hardware and software setups and differing measures of efficiency (Dehghani et al., 2021). An upper bound on the compute of a training run can be established by finding the total number of (low-precision) floating point operations available over the wallclock budget of the run. This peak of total FLOPs in a given time interval is generally not reached in actual compute, even for highly optimized models (Chowdhery et al., 2022), but represents the paid budget required to realize a training run. We summarize budgets for a few select training runs in Table 1. After the original training run for BERT on TPUs, initial reactions estimated up to 11 days of compute for comparable results on GPUs (Dettmers, 2018). However, sustained improvements, especially in software, have reduced the upper limit significantly (You et al., 2019; Narasimhan, 2019). Yet, recipes and implementations generally require entire server nodes (for GPUs) or TPU slices and target larger BERT architectures.

The only case where I am willing to accept FLOPS being used as some metric

Other work discussing improvements to BERT targets compute settings closer to the original BERT, for example SqueezeBERT (Iandola et al., 2020) employs 8 Titan RTX cards for four days. Sellam et al. (2022) note that the original BERT training run is an outlier and doubling its training time more reliably reproduces the original results.

Our central point of comparison for BERT training with limited resources is the work of Izsak et al. (2021) who also attempt the goal of training BERT within 24 hours with overall similar limitations, but use a full server node with 8 V100 GPUs. Izsak et al. (2021) choose a BERT$_{\text{LARGE}}$ architecture variant and train with sequence length of 128, including a range of tweaks such as modified learning rates schedules, large batch sizes, sparse prediction and packed sequences. We re-evaluate this setup as a baseline setting for our own compute budget (which is about 15x smaller).

Studies of Efficient Transformers Recent years have seen a flurry of research working to improve and modify the transformer architecture proposed in Vaswani et al. (2017) and we refer to Treviso et al. (2022) for a recent categorization and review of research in this area. Several meta-studies have investigated proposed improvements and modifications: Narang et al. (2021) evaluate a large range of architectural modifications applied to the T5 model pipeline of Raffel et al. (2020) on tasks in both language understanding and translation. The encoder-decoder structure of T5 is closer in spirit to the original transformer setup, but is understood to behave similarly to BERT when using the encoder component (Liu et al., 2021a). Evaluating modifications with 1.75 days of compute on `TPU` slices they find that most improvements do not reliably materialize gains in final accuracy. Tay et al. (2021) work in the same setting and evaluate the optimal shape of T5 derived architectures and its relative effects on downstream performance as models are scaled. Further exploration of the scaling behavior of various architectural improvements in Tay et al. (2022a) find that only few modifications outperform the original architecture of Vaswani et al. (2017) at all scales, especially when evaluating downstream accuracy. The meta-study investigating improvements in preparation for extreme-scale training in Scao et al. (2022) focuses on minor modifications to layout, positional embeddings and data sources for autoregressive models, and other extremely-large scale training runs have so far been similarly conservative in their settings (Brown et al., 2020; Black et al., 2022; Rae et al., 2022).

In general though, these evaluations target larger compute settings than we intend to use, and are concerned with whether improvements (often from academic sources and proposed with evaluations on small scales) translate to larger scales. In this work, we set aside the question of (up)scaling and focus only on the limited compute.

What factors affect the performance most when scaling transformers?

Scaling Laws The difficulty in finding tangible improvements is echoed in the scaling laws of Kaplan et al. (2020). Over a wide range of transformer model shapes, Kaplan et al. (2020) find only model size (as number of parameters in non-embedding layers) strongly predicts performance. Further, for a fixed compute budget, an optimal model size can be derived, but performance is only mildly connected to model size - larger models processes less data per unit of compute, but improve faster by almost the same margin. While the precise coefficients and shape of these scaling laws continue to be iterated on (Hoffmann et al., 2022) and adapted for related settings (Bansal et al., 2022; Clark et al., 2022; Bahri et al., 2021b), their overall logic appears hard to escape, even if power laws fit observations somewhat less well on small scales.

4 Investigations

For our experimental evaluation we implement and test a considerable number of proposed modifications to the setup of Devlin et al. (2019) for their merits in our limited compute setting as described in Section 2. We first clarify the common implementation and initial data setup, and then investigate architectural, training and dataset improvements.

4.1 Implementation Details

We implement everything in PyTorch (Paszke et al., 2017) and to limit our gains from the ”software lottery” (Hooker, 2021) we do not use specialized implementations, which would further bias results towards well-established components. We keep everything on the implementation level of the PyTorch framework, allowing only automated operator fusion (Sarofeen et al., 2022) that can be applied to all components. Only after choosing a final architecture variant, we then re-enable the efficient attention kernel described in Dao et al. (2022). We run all experiments and ablation studies with the same setup of automated mixed precision (Micikevicius et al., 2018) for standard 16- and 32-bit floating point precision (over full 32-bit float, scaled 16-bit (Rasley et al., 2020) and pure `bfloat16` (Wang & Kanwar, 2019). We find no benefit from offloading (Ren et al., 2021; Rasley et al., 2020) in our setting.).

Initial Data Setup We start our investigation with a close analogue to the original raw text sources of Devlin et al. (2019), using a recent dump of the English Wikipedia (`20220301.en`) and En-

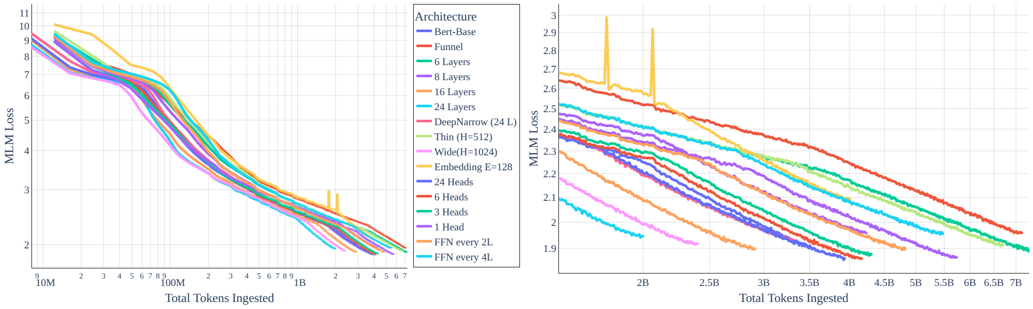

Figure 1: Various Transformer architectures and shapes, showing MLM loss versus number of tokens ingested. Left: Global view. Right: Zoom onto 10e8 or more tokens. All models trained with the same budget. We see that improvements through architectural reshaping are minimal; while there are some fluctuations in loss early in training, the rates of loss decay during most of training differ by a multiplicative constant (horizontal shift due to logarithmic horizontal axis) that depends strongly on the model size and not model type.

glish `bookcorpus`, noting the commentary of Tan (2019); Bandy & Vincent (2021). We force all text into lower-case, strip accents and non-ascii characters and create an English tokenizer from scratch based only on this data. We choose WordPiece with a vocabulary size of $2^{15} = 32768$ (Wu et al., 2016). We found no significant change in performance with BPE (Sennrich et al., 2016) or SentencePiece with Unigrams (Kudo, 2018; Kudo & Richardson, 2019). Smaller vocabulary sizes ($2^{12}, 2^{13}, 2^{14}$) resulted in worse performance, while larger vocabulary sizes (2^{16}) we not reliably better. We pack tokenized data into randomized sequences of length 128 and separate unrelated fragments by `<sep>`. The performance impact from dropping this separator was minimal. No impact was observed from including a `<cls>` token in pretraining. The shorter sequence length is sufficient for the downstream applications that we are targeting and simplifies attention computations. Packing data into full sequences limits us to simpler sequence losses, but uses the available compute optimally Liu et al. (2019); Izsak et al. (2021). For the targeted compute settings, this sequence length results in micro-batch sizes of 64 to 96 for most variations of the base BERT architecture on the `gtx2080ti`, which we will accumulate into larger batch sizes. With our limited compute budget, this produces enough samples to run single-epoch training (Komatsuzaki, 2019; Hernandez et al., 2022) where no data point is revisited.

4.2 Modifying the Architecture

The most obvious way to efficiently scale down training is by modifying the model architecture; intuitively, it seems likely that smaller/lower capacity models will be optimal in the cramming regime. In this section, we study the relationship between model type and training efficiency. We see that scaling laws create a strong barrier to scaling down. Per-token efficiency of training depends strongly on model size, but not transformer type. Furthermore, smaller models learn less efficiently, and this largely mitigates any throughput gains. Fortunately, the fact that training efficiency is nearly constant across models of the same size means that we can boost performance by finding architecture modifications that speed up gradient computation while keeping the parameter count nearly constant. This makes architecture selection fairly straightforward as we can make design choices based primarily on how they affect computation time for a single gradient step.

Scaling laws hold in the low-resource regime A large corpus of research in recent years has developed architectural improvements to speed up the original transformer. Many of these methods have not been found to improve training for the large-scale T5 architecture Narang et al. (2021); Tay et al. (2022a). But, in the low compute setting where data throughput is of utmost importance, maybe this is the way forward? Scaling laws have been observed by Kaplan et al. (2020) in the high-resource regime, and seem to hold strongly in the limit as resources grow. Surprisingly, these laws also hold in the limit of extreme compute down-scaling, and they create a barrier to low-cost training.

We exemplify the effect of scaling laws for many transformer variants from the literature in Figure 1, where we train each architecture variant with optimized training hyperparameters as described below in Section 4.3. We apply these architecture variants to a shared baseline model that incorporates

Pre-Normalization and rotary embedding. Figure 1 visualizes the progress of MLM loss versus the number of tokens ingested in total and all architectures run with the same time budget.

We observe that varying the transformer type and size has only minimal impact on the final loss after 24 hours. Models with more parameters learn more efficiently, as their MLM loss decreases faster on a per-gradient basis. However, smaller architectures make up for their slower learning efficiency by higher throughput, and thus process more tokens over the limited budget. Figure 1 shows that different architectures are unpredictable throughout an initial stage of training (the first 1B tokens), after which the per-token efficiencies differ by only a multiplicative constant (a horizontal shift due to the log axis). This constant depends almost entirely on the model size, not model type, so that all choices reach a MLM loss around 1.9 at the end of training.

Exploiting the scaling law. The scaling laws seem to bar us from making large gains via major changes to the transformer size and type, as per-token performance is tightly coupled to model size. As a result, we find no improvements when using a funnel-transformer architecture (Dai et al., 2020; Nawrot et al., 2022), when dropping FFN layers (Sridhar et al., 2022), or when using recurrent layers (Lan et al., 2019), even when trained with BPTT as in Schwarzschild (2021). Rescaling architectures to be deep-narrow (Tay et al., 2021; Wies et al., 2021) provides no gains.

While this principle closes one door for scaling down efficiently, it opens another; Because per-gradient efficiency remains nearly constant for all models of the same size, we can exploit scaling laws by quickly searching for architectural choices that speed up computation while keeping model size roughly constant. A number of obvious optimizations fall into this category, and we describe them below, in addition to several other tweaks that provide marginal but worthwhile/free gains.

Attention Block: We disable all QKV biases (Dayma et al., 2021). This exploits the scaling law by removing a layer of computation, making the forward and backward pass somewhat faster, while keeping the model size nearly constant. We find that we could decrease gradient costs by reducing the number of attention heads (Merity, 2019; Araabi & Monz, 2020; Liu et al., 2021b; Javaheripi et al., 2022), as this parallelizes better on the GPU and provides a slight performance boost. Yet, reducing the amount of heads also decreases finetuning performance, so we ultimately keep all 12 heads. We find no benefits from replacements to the softmax operation (Richter & Wattenhofer, 2020). We further keep the original multi-head self-attention mechanism. A large amount of work has been focused on efficient attention (Sukhbaatar et al., 2019; Beltagy et al., 2020; Wang et al., 2020a; Liu et al., 2021c) and studies of efficient attention (Tay et al., 2020a;b). But, because we set the maximal sequence length to 128, attention complexity is less of a concern in our setting. To verify this, we implement the recently proposed FLASH mechanism (Hua et al., 2022), but find no benefits. We further experiment with Fourier attention as proposed in Lee-Thorp et al. (2021), but find no improvements. We find rotary embeddings (Su et al., 2021; Black et al., 2022), to provide small benefits, but these are evened out by the drop in speed, so we ultimately decide against these.

Feedforward Block: We find empirical gains from disabling all linear layer biases (Dayma et al., 2021). Just as for the attention layers, this leverages the scaling law by accelerating gradient computation without noticeable impacts on model size. As a result, we get higher throughput without compromising the rate at which the model improves. We keep the original feedforward block largely unchanged, finding no benefits from changing to another activation than GELU. We do see small improvements from re-ordering the block into a gated linear unit (Dauphin et al., 2017). In contrast to other work, e.g. (Black et al., 2022), we do not increase the number of parameters in the FFN block to compensate for the halving of the hidden dimensionality due to gating.

Embedding: We implement scaled sinusoidal positional embeddings as described in Hua et al. (2022), finding incremental benefits over learned or unscaled sinusoidal embeddings. We see no improvements from decoupling the input and output embeddings (Chung et al., 2020). The suggestion from Lan et al. (2019) to factorize the input embedding provides no gains in our setting. We include a layer normalization at the end of the embedding block.

Layer Structure: As observed in many studies, we find that pre-normalization with Layer Norms is beneficial over post Layer Norms (Baevski & Auli, 2018; Xiong et al., 2020). We see no additional benefit from other variants of this modification, such as (Liu et al., 2020b; Shleifer et al.,

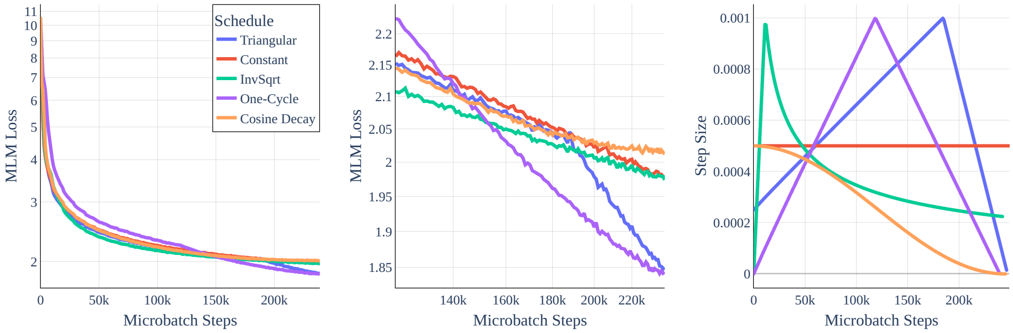

Figure 2: Learning Rate Schedules. Although globally many schedule result in similar behavior, we see in the zoom in the middle, that differences do exist. The right side shows the corresponding learning rate schedules. Both triangular-shaped one-cycle schedules have better end-time behavior, possibly due to the quick annealing.

2021). Further, replacing Layer Normalization with RMS Normalization provides no gains (Zhang & Sennrich, 2019). We note that the key effect of pre-normalization is to stabilize training and enable larger learning rates and reduced warmup, and we see limited benefits from including it by itself. We see no benefits from stochastic dropping of entire layers as described in (Zhang & He, 2020).

Head Block: We find that we can remove the nonlinear head without ill effect. We can further drop the decoder bias (Radford et al., 2019) and gain in memory using sparse token prediction (Liu et al., 2019; Izsak et al., 2021). We add a final Layer Norm to stabilize training further.

4.3 Modifying the Training Setup

We study the impact of training hyper-parameters on the BERT-base architecture. The original BERT training recipe understandably results is poor model performance in the cramming setting, and so we revisit a number of standard choices.

Objective: We train with only masked language modeling on fully packed blocks of tokens with a masking rate of 15% and the original setup of Devlin et al. (2019) where 10% of all masks are filled with random words and 10% unchanged. We see no improvement from masking at larger rates, e.g. at 40% as proposed in (Wettig et al., 2022), see Appendix. We see no difference enabling or disabling the mentioned 20% rule. We evaluate other functions for the masked-language objective, such as mean-squared error (Hui & Belkin, 2021) or L1 loss, but find no benefits.

Choice of Optimizer: We keep Adam (Kingma & Ba, 2015) as the optimizer of choice, with weight decay of 0.01 as described in (Loshchilov & Hutter, 2017), $\beta_1 = 0.9, \beta_2 = 0.98$ and $\varepsilon = 10^{-12}$. To stablize training at no extra cost, we include gradient clipping at a clip value of 0.5. We find no noticeable change in varying these parameters in reasonable amounts, e.g. $\varepsilon = 10^{-6}$, $\beta_1 = 0.9, \beta_2 = 0.999$. We test other first-order adaptive optimizers (Shazeer & Stern, 2018; Liu et al., 2020a) but find no advantages in our setting. We further find no advantages using higher-order optimizers (Yadav, 2020; Anil et al., 2021), but note that especially for higher-order optimizers there is a greater amount of variability in implementation.

Learning Rate Schedule and Peak: Following the advice of Izsak et al. (2021), we re-scale the learning rate schedule so that it is tied to our budget and the learning rate decays as the budget reduces to zero. Interestingly, we observe in Figure 2 that while globally a large number of learning rate shapes lead to similar reductions in loss, we find that we can make some gains through the choice of schedule. We find that a simple one-cycle learning rate (Smith & Topin, 2018) with a peak learning rate of 10^{-3} leads to minimal pretraining loss within our budget.

Batch Size Schedule: A particularity of our setting is that, due to being limited to a single GPU, the micro-batch size that finds its way onto this GPU (96 for most experiments) is several times smaller than the optimal batch size. We find that the optimal batch size in this setting is around

Dataset	Batch Size	MNLI (m)	
Bookcorpus-Wikipedia	1536	79.8	
The Pile	1536	80.5	
The Pile (natural data subset)	1536	80.8	
C4-Subset	1536	79.1	①
Bookcorpus-Wikipedia, Deduduplication > 100	1536	79.9	
Bookcorpus-Wikipedia, Deduduplication > 50	1536	79.5	
Bookcorpus-Wikipedia, filtered with $t = 0.3$, sorted	1536	80.8	
Bookcorpus-Wikipedia, sorted	1536	81.0	
C4-Subset, Deduduplication > 100	1536	79.2	②
C4-Subset, filtered with $t = 0.3$	1536	79.9	③
C4-Subset, filtered with $t = 0.3$, sorted	1536	81.4	
C4-Subset, filtered with $t = 0.3$, larger, sorted	1536	81.9	
Bookcorpus-Wikipedia	4032	80.5	
C4-Subset, filtered with $t = 0.3$	4032	82.2	
C4-Subset, filtered with $t = 0.3$, sorted	4032	82.5	④
C4-Subset, filtered with $t = 0.3$	8064	80.9	

Table 2: Dataset Variations for the optimal model from Section 4.2 and optimal training routine from Section 4.3, modifying final batch size in conjunction with dataset format.

1536 for minimal pretraining loss, but 4032 for maximal downstream performance for the 2080ti, i.e. we accumulate gradients and only perform an update every 16 and 42 forward/backward passes, respectively. For the larger A4000 and A6000 cards, this corresponds to a micro-batch size of 128/256 and final batch size of 4096, which we again accumulate.

Fortunately, we can find small speedups by using an aggressive batch size schedule; we increase the number of averaged micro-batches linearly over the course of training. This results in more progress earlier in training, and leads to a small benefit to performance. We also experiment with automatic and adaptive batching rules (De et al., 2017; Bollapragada et al., 2018a;b), but find that the best results from these adaptive schedules resemble the fixed linear schedule. For simplicity we just stick to the simpler linear schedule.

Dropping Dropout The original BERT model of Devlin et al. (2019) includes dropout as in Vaswani et al. (2017), which prevents overfitting when training data is small relative to total compute budget. While it can be helpful as a regularizer, dropout effectively reduces the number of gradient updates seen by each parameter, as updates do not occur when the associated feature is dropped. At the same time, update runtime is not strongly effected by the presence of dropout, and so dropout results in a net reduction in updates per second.

In the cramming setting, training data is large compared to compute. Overfitting is not possible due to the single epoch schedule, and we disable dropout during pretraining (Brown et al., 2020) to maximize the number of parameter updates. We re-enable dropout during downstream fine-tuning with a dropout value of 0.1. Further, we experiment with length curricula (Li et al., 2022) (see appendix) and token dropping (Hou et al., 2022), but find no gains in our setting.

4.4 Optimizing the Dataset

We found above that scaling laws create a barrier to making major gains (beyond computational efficiencies) with architectural modifications. However, scaling laws do not preclude us from training on better data. Once we have exhausted our ability to train on more tokens per second, we should seek to train on better tokens.

We consider two data based pathways to better down-scaling. First, we can filter, process, or sort the existing data in various ways. Second, we can swap our data source. To this end, we experiment with several subsets of *The Pile* (Gao et al., 2020), containing raw text from only *Gutenberg*, *Books3* and *Wikipedia (en)*. From these Pile datasets we tokenize the first 4×10^6 entries to generate enough tokens for our single pass. Another popular source of data is C4, the colossal, cleaned version of

Common Crawl (Raffel et al., 2020), from which we stream the first 20×10^6 entries. For each data source we regenerate its own WordPiece tokenizer as described in Section 4.1.

Of these four sources, we find the Pile to perform best in terms of downstream MNLI performance. However, it turns out we can further improve especially the C4 datset through additional processing. We first evaluate deduplication as described in Lee et al. (2022) via exact substring deduplication, but find this not to help in downstream performance in our case. We then test filtering for uncompressible data. We use the tokenizer itself to remove all training sequences from C4 set that cannot be compressed well; we simply set a threshold t, e.g. $t = 0.3$, and drop all entries from the dataset where the number of tokens in the entry is larger than t times the number of raw characters. This removes, for example, sequences consisting of hard-to-compress HTML or markdown code. Surprisingly, this results in a measurable improvement on C4, summarized in Table 2.

We then see some further improvements from two directions. First, sorting all tokenized sequences by some metric, and second, increasing the final batch size. For filtering we sort all tokenized sequences by their average (unigram) token prevalence, so that likely sequences occur first. This has some positive effect, and can be strengthened slightly by drawing from a larger corpus, as the unlikely sequences never get reached. Finally, increasing the batch size to $4032/4096$ at the end of training (as mentioned in Section 4.3) is disproportionally effective on C4, but less so on `bookcorpus-wikipedia`. We believe that both modifications ultimately reduce the likelihood of training being hindered by fluctuations in the data distribution.

Vocabulary Size We also check whether the original vocabulary size of 32768 described in (Devlin et al., 2019) is optimal in the crammed regime. A priori, this might not hold: The larger, the vocabulary, the more, unique tokens and relationships between unique tokens have to be learned during training. On the other hand, increasing the vocabulary size would compress data further (albeit vanishingly after some point), which would allow for more information to be compressed into the fixed number of tokens that can be ingested during the crammed training run. In Figure 3, we find that for `bookcorpus-wikipedia` data, larger vocabulary sizes correlate with larger average GLUE score, although the effect is plateauing for the MNLI task around the original 32768 vocabulary size. Moving forward, we hence keep this vocabulary size.

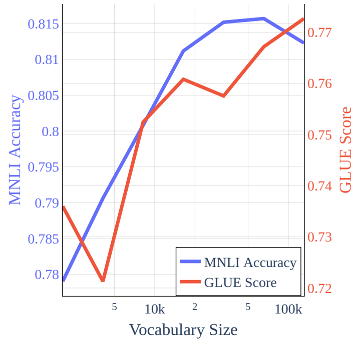

Figure 3: Vocabulary Size versus GLUE Score and MNLI Accuracy for models trained in the cramming regime on `bookcorpus-wikipedia` data.

5 Finetuning Performance on GLUE

Finally, we systematically evaluate performance on the GLUE benchmark of Wang et al. (2018), minus WNLI as in Devlin et al. (2019). We note that we only use MNLI (m) during the previous sections and do not tune hyperparameters based on the full GLUE scores. We finetune both the pretrained BERT-base checkpoint and our models under the same constraints laid out in Section 2. For BERT-base, we finetune all datasets for 5 epochs with a batch size of 32 and learning rate of 2×10^{-5}. For the crammed models, we find that this is not optimal and minor improvements can be gained from a batch size of 16 and learning rate of 4×10^{-5} with cosine decay (this setup does not improve the pretrained BERT checkpoint).

Table 3 and Table 4 describe the performance of this setup on the GLUE downstream tasks (as median over 5 downstream trials). There we compare the original BERT-base checkpoint, a reproduction of the BERT pretraining settings stopped after our budget is reached, the setup described in (Izsak et al., 2021) and the modified recipe, trained for a single day for each GPU setup. Overall, performance is surprisingly decent, especially for the larger datasets of MNLI, QQP, QNLI and SST-2, where downstream finetuning can smooth out the remaining differences between the full BERT model and the crammed variants. Further, we find substantial gains over both a naive BERT

	MNLI	SST-2	STSB	RTE	QNLI	QQP	MRPC	CoLA	GLUE
BERT-base (Fully trained)	83.2/83.4	91.9	**86.7**	**59.2**	**90.6**	**87.7**	**89.3**	**56.5**	**80.9**
BERT-base (No Pretrain)	34.1/34.1	79.9	17.8	47.3	50.0	68.6	77.9	0.0	45.5
Trained for 1 day on a **2080ti**:									
BERT (normal protocol)	58.7/57.8	79.8	16.6	50.9	55.4	71.1	70.1	7.3	52.0
BERT ((Izsak et al., 2021))	75.0/75.7	-	-	52.3	84.6	84.4	82.2	33.8	69.7
crammed BERT	82.8/83.4	91.5	83.1	54.0	89.0	87.2	86.2	47.2	78.3
Trained for 1 day on an **A4000**:									
BERT (normal protocol)	58.0/56.5	79.4	17.0	51.6	54.2	70.6	74.1	8.2	52.2
BERT ((Izsak et al., 2021))	58.8/59.6	-	-	-	-	-	81.4	0.0	49.9
crammed BERT	83.0/83.2	91.6	84.8	54.7	88.5	86.9	86.4	43.7	78.1
Trained for 1 day on an **A6000**:									
BERT (normal protocol)	56.3/54.8	81.2	21.8	49.5	56.4	65.1	74.8	10.3	52.2
BERT ((Izsak et al., 2021))	76.2/76.5	87.4	78.5	49.1	85.0	84.1	83.2	36.3	72.9
crammed BERT	**83.9/84.1**	**92.2**	84.6	53.8	89.5	87.3	87.5	44.5	78.6

Table 3: Comparison in GLUE-dev performance of baseline BERT to the crammed model. Note that all runs abide by the finetuning protocol described in Section 2 with fixed hyperparameters for all tasks and an epoch limit of 5. Missing values are NaN. The protocol of (Izsak et al., 2021) was designed for an 8 GPU server blade, and it crammed onto a single GPU here. The MNLI column shows evaluation results for both matched and mismatched sets. The GLUE column depicts the full average over the same tasks as in Devlin et al. (2019).

training with limited budget, and over the recipe described in (Izsak et al., 2021). For (Izsak et al., 2021), the described recipe was originally designed for a full 8 GPU server blade, and squeezing the BERT-large model therein onto the smaller GPUs in this experiment is resposnsible for most of the performance degradation of this recipe in our scenario.

Overall, the crammed model mostly works, even for smaller datasets. The average is brought down however by a significant drop on CoLA (corpus of linguistic acceptability) (Warstadt et al., 2019). This behavior is intriguing and we offer two hypotheses. First, it is conceivable that the chosen global hyperparameters for finetuning are a bad fit for CoLA in particular. CoLa performance can be brittle with respect to hyperparameter, with Jiao et al. (2020) training longer only on CoLA or Joshi et al. (2020) training less only on CoLA. Nevertheless, for BERT, a set of global hyperparameters exists, pointing at a deficiency in the crammed model. As a second hypothesis, it is conceivable that these models need to process more text before they memorize enough data to do well on CoLA. This would be in contrast to Liu et al. (2021d), who find that CoLA is learned relatively quickly compared to other downstream tasks when probing intermediate BERT checkpoints. On the other hand, deficiencies on CoLA in particular are also common in approaches that distill BERT into smaller architectures (Sun et al., 2019; Turc et al., 2019; Mukherjee et al., 2021), which might come with limited capacity for linguistic acceptability.

Table 4: Comparison in GLUE-dev performance of baseline BERT to crammed model. Avg. Score is all scores excluding CoLA, GLUE is the full average over the same tasks as in Devlin et al. (2019).

	CoLA	Avg. Score	GLUE
Bert-Base	**56.5**	**84.0**	**80.9**
Crammed (2080ti)	47.2	82.1	78.3
Crammed (A4000)	43.7	82.4	78.1
Crammed (A6000)	44.5	82.9	78.6

5.1 Ablation - Which Changes Really Mattered?

In Table 5 we provide a summary ablation study of all changes discussed in this work. We group modifications, as in previous sections into the three groups of architecture, training and data and ablate each group by resetting all modifications to the original BERT recipe. Here, we find that we first have to make minimal modifications in any case, as modifications to architecture, such as PreNorm layer structures also in turn allow the more aggressive learning rate schedules described in the training setup. Taking this into account, we ultimately find about two percentage points gained in average GLUE score through architectural changes, one percentage point in data changes, and half a precentage point in training modifications.

	MNLI	SST-2	STSB	RTE	QNLI	QQP	MRPC	CoLA	GLUE
crammed BERT	**83.9 / 84.1**	**92.2**	84.6	53.8	**89.5**	**87.3**	87.5	**44.5**	**78.6**
+ original data	82.2 / 82.7	92.0	83.6	49.8	**89.5**	87.0	85.9	42.5	77.3
+ original train	50.0 / 50.4	80.7	13.7	52.0	59.8	65.1	73.2	7.2	50.2
+ original arch.	35.4 / 35.2	49.1	-	52.7	49.5	0.0	0.0	0.0	27.7
+ minimal train mod.	81.9 / 82.6	91.4	**85.5**	**54.9**	88.2	87.0	**88.4**	43.6	78.1
+ minimal arch. mod.	83.2 / 83.5	91.7	82.0	52.0	88.9	86.8	83.6	38.3	76.7

Table 5: Ablation study, which improvements were most important? The first group shows an ablation where one component of the final combination of training, architecture, and data modifications (the crammed BERT model) is replaced by the original setup. Here, we find that modifications in training and architecture have to co-occur. For example, the aggressive learning rate schedule can only be used when the model also contains pre-normalization Layer Norms. As such we also include a row with *minimal training* modifications (dropout disabled, cosine decay to zero within budget with warmup, fixed batch size of 4096) and a row with *minimal architecture* modifications (Pre-normalization, sparse activations, Layer Norm $\varepsilon = 10^{-6}$).

	MNLI	SST-2	STSB	RTE	QNLI	QQP	MRPC	CoLA	GLUE
BERT-Base (Fully trained)	83.2/83.4	91.9	86.7	59.2	90.6	87.7	89.3	56.5	80.9
BERT-Base (No Pretrain)	34.1/34.1	79.9	17.8	47.3	50.0	68.6	77.9	0.0	45.5
ROBERTA-Base	**86.6**/86.4	93.7	**90.4**	**77.3**	92.1	**88.3**	**91.4**	**60.2**	**85.1**
Crammed BERT (A6000)	83.9/84.1	92.2	84.6	53.8	89.5	87.3	87.5	44.5	78.6
Trained for 2 days on 8 **A6000**:									
Crammed BERT	86.5/**86.7**	**93.8**	86.8	53.4	91.6	88.0	88.2	42.9	79.8
Crammed BERT (no clipping)	86.1/**86.7**	93.2	87.1	55.2	**92.1**	88.3	90.2	46.6	80.6

Table 6: Models trained on 16x as much compute as otherwise in this work, but with exactly the same setup and data. With this budget, we use about half as much compute as one of the original BERT training runs. The resulting models (which are surprisingly slightly improved by removing gradient clipping again), are equivalent in performance, even to ROBERTA-base models trained in Liu et al. (2019), on some tasks. On other tasks, such as CoLA and RTE, the additional compute barely improves performance.

5.2 What Happens When Training Longer?

We also verify what happens if the cramming recipe discussed so far is used with more budget. To this end, we train models for 48 hours on 8 A6000 GPUs, which ends up to be 208 total exaFLOP, c.f. Table 1. We directly apply the setting described so far, simply scaling the learning rate schedules to cover the new budget of 48 hours. In Table 6, we find that the discussed recipe does immediately generalize to larger compute budgets. This is surprising, not the least, as now, the dataset (which was sorted in Section 4.4 is now too small and repeated multiple times. The newly trained models have strong performances, especially on MNLI and SST-2, where they significantly outperform the original BERT checkpoint and fall into a similar range as the roBERTA-base checkpoint of Liu et al. (2019), which was trained with much more compute. Yet, in other tasks, such as (again) CoLA, the new models barely improve even in the larger compute regime.

6 Limitations

In this work, we limited our investigation to transformer-based architectures trained with MLM objectives. However, we do think that the general task of cramming posed in Section 2 is interesting even when relaxing these constraints. There have been a number of modifications proposed to the objective in particular (Joshi et al., 2020; Bao et al., 2020; Bajaj et al., 2022; Tay et al., 2022b). While Artetxe et al. (2022) and Wang et al. (2022) find MLM still to hold up well as a pretraining objective, other suggestions such as ELECTRA (Clark et al., 2019; 2020; He et al., 2021) could be employed which might be beneficial for crammed models. Also, the optimal architecture might not be transformer-based (Merity, 2019; Fusco et al., 2022; Peng, 2021).

7 Conclusions

We discuss how much performance a transformer-based language model can achieve when crammed into a setting with very limited compute, finding that several strands of modification lead to decent downstream performance on GLUE. Overall though, cramming language models appears hard, as we empirically find many implications of Kaplan et al. (2020) to still hold in this regime, and for examples improvements through larger models are evened out by their slower speed. We hope that this work can provide a baseline for explorations of the question of cramming we formalize in Section 2 and cast an additional light on a number of improvements and tricks proposed for transformer architectures in recent years.

Reproducibility Statement

We provide code to reproduce all experiments at https://github.com/JonasGeiping/cramming.

References

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Scalable Second Order Optimization for Deep Learning. *arXiv:2002.09018 [cs, math, stat]*, March 2021. URL http://arxiv.org/abs/2002.09018.

Ali Araabi and Christof Monz. Optimizing Transformer for Low-Resource Neural Machine Translation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 3429–3435, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.304. URL https://aclanthology.org/2020.coling-main.304.

Mikel Artetxe, Jingfei Du, Naman Goyal, Luke Zettlemoyer, and Ves Stoyanov. On the Role of Bidirectionality in Language Model Pre-Training. *arxiv:2205.11726[cs]*, May 2022. doi: 10.48550/arXiv.2205.11726. URL http://arxiv.org/abs/2205.11726.

Alexei Baevski and Michael Auli. Adaptive Input Representations for Neural Language Modeling. In *International Conference on Learning Representations*, September 2018. URL https://openreview.net/forum?id=ByxZX20qFQ.

Dara Bahri, Hossein Mobahi, and Yi Tay. Sharpness-Aware Minimization Improves Language Model Generalization. *arXiv:2110.08529 [cs]*, October 2021a. URL http://arxiv.org/abs/2110.08529.

Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining Neural Scaling Laws. *arxiv:2102.06701[cond-mat, stat]*, February 2021b. doi: 10.48550/arXiv.2102.06701. URL http://arxiv.org/abs/2102.06701.

Payal Bajaj, Chenyan Xiong, Guolin Ke, Xiaodong Liu, Di He, Saurabh Tiwary, Tie-Yan Liu, Paul Bennett, Xia Song, and Jianfeng Gao. METRO: Efficient Denoising Pretraining of Large Scale Autoencoding Language Models with Model Generated Signals. *arXiv:2204.06644 [cs]*, April 2022. URL http://arxiv.org/abs/2204.06644.

Jack Bandy and Nicholas Vincent. Addressing ”Documentation Debt” in Machine Learning: A Retrospective Datasheet for BookCorpus. *NeurIPS 2021 Track Datasets and Benchmarks*, November 2021. URL https://openreview.net/forum?id=Qd_eU1wvJeu.

Yamini Bansal, Behrooz Ghorbani, Ankush Garg, Biao Zhang, Maxim Krikun, Colin Cherry, Behnam Neyshabur, and Orhan Firat. Data Scaling Laws in NMT: The Effect of Noise and Architecture. *arXiv:2202.01994 [cs]*, February 2022. URL https://arxiv.org/abs/2202.01994v1.

Hangbo Bao, Li Dong, Furu Wei, Wenhui Wang, Nan Yang, Xiaodong Liu, Yu Wang, Songhao Piao, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. UniLMv2: Pseudo-Masked Language Models for Unified Language Model Pre-Training. *arXiv:2002.12804 [cs]*, February 2020. URL http://arxiv.org/abs/2002.12804.

Summary

The last few years have shown tremendous success in performing large language models at scale. By scaling transformers, we have seen ground-breaking improvements in many sub-fields of natural language processing, including language understanding and natural language generation.
Though scaling is good for transformers models, this trend has resulted in an environment where training language models is out of reach for most researchers and practitioners. The authors take a different approach here and see how far they can get in training a transformer-based model from scratch on a single consumer GPU in a single day.

Constraints

- Transformer-based language models of arbitrary size are trained completely from scratch.
- No existing pretrained models in any part of the pipeline.
- Any raw text (excluding downstream data) can be included for training
- The downloading and pre-processing of raw data are exempted from the total compute budget. Pre-processing may include CPU-based tokenizer construction, tokenization, and filtering, but cannot include representation learning (e.g. pre-training a word embedding is not allowed unless it is counted towards the final runtime).
- Training proceeds on a single GPU for 24 hours (rtx2080ti/rtxa4000/rtxa6000)
- Downstream performance is evaluated on GLUE Downstream finetuning is excluded from the total compute budget.

To start with one might ask the fundamental question: How long does it take to train BERT?
Sadly, the answer to this question isn't straightforward as the efficiency depends on the hardware and software setup used. An upper bound on the compute of a training run can be established by finding the total number of (low-precision) floating point operations available over the wallclock budget of the run. This peak of total FLOPs in a given time interval is generally not reached in actual compute, even for highly optimized models, but represents the paid budget required to realize a training run. It is known that model size (as the number of parameters in non-embedding layers) strongly predicts performance. Further, for a fixed compute budget, an optimal model size can be derived, but performance is only mildly connected to model size. Larger models process less data per unit of compute but improve faster by almost the same margin

Data

Setup:

- ○ Recent dump of English Wiki and English bookcorpus
- ○ All text into lower-case, strip accents, and non-ASCII characters
- ○ Wordpiece tokenizer with vocab size of 2ˆ15 = 32768
- ○ Tokenized data into randomized sequences of length 128 separated by a <sep> token

Findings:

- ○ No significant change in performance with BPE or SentencePiece or Unigrams
- ○ Smaller vocabulary sizes (2ˆ12, 2ˆ13, 2ˆ14) resulted in worse performance, while larger vocabulary sizes (2ˆ16) we not reliably
- ○ better.
- ○ Including <cls> token has no impact on pretraining

Architecture

Setup

- Per-token efficiency of training depends strongly
- on model size, but not transformer type
- Training efficiency is nearly constant across models of the same size means that we can boost performance by finding architecture modifications that speed up gradient computation while keeping the parameter count nearly constant
- Disable all QKV biases making forward and backward pass somewhat faster
- All 12 attention heads are kept as it is
- The Feedforward block is kept as it is
- Scaled sinusoidal embedding. Layer normalization at the end of the embedding block.
- PreNormalization instead of post-normalization

Findings:

- Varying the transformer type and size has only minimal impact on the final loss after 24 hours
- Models with more parameters learn more efficiently, as their MLM loss decreases faster on a per-gradient basis. However, smaller architectures make up for their slower learning efficiency by higher throughput and thus process more tokens over the limited budget
- Different architectures are unpredictable throughout an initial stage of training (the first 1B tokens) after which the per-token efficiencies differ by only a multiplicative constant (a horizontal shift due
- to the log axis). This constant depends almost entirely on the model size, not the model type so that all choices reach a loss value of around 1.9 at the end of the training
- No improvements using funnel transformers or dropping FFN layers
- No benefits from replacements to the softmax operation in the attention mechanism
- No benefits were observed from using efficient attention mechanisms like FLASH-attention
- No major benefits from special embeddings like rotatory embeddings
- No benefits were observed by replacing GELU with some other activation
- No benefits were observed by factorizing the embeddings
- Prenormalization only provides the benefit of stable training

Training

Setup:

- A 15% masking rate is used
- Adam as the optimizer with weight decay of 0.01, b1=0.9, b2=0.98, and epsilon=10^-12
- Gradient clipping at a value of 0.5 for stable training
- Learning rate with a schedule, one cycle with a peak of 10^-3 in this case.
- Batch size of 96 with a linear batch size schedule
- No dropout during pretraining but enabled for downstream fine-tuning

Findings:

- No improvements from large masking rates
- No improvements from using a different loss function

Extra insights:

- To train on better tokens, we can do two things. First, we can filter, process, or sort the existing data in various ways. Second, we can swap our data source
- For filtering the authors sort all tokenized sequences by their average (unigram) token prevalence so that likely sequences occur first. This has some positive effects and can be strengthened slightly by drawing from a larger corpus, as the unlikely sequences never get reached.
- GLUE score increases with an increase in the vocabulary size but the gains plateau around the original 32768 vocabulary size
- Overall, the crammed model mostly works, even for smaller datasets. The average is brought down however by a significant drop on CoLA
- The authors find about two percentage points gained
- in average GLUE score through architectural changes, one percentage point in data changes, and half a percentage point in training modifications.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The Long-Document Transformer. *arXiv:2004.05150 [cs]*, December 2020. URL http://arxiv.org/abs/2004.05150.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. GPT-NeoX-20B: An Open-Source Autoregressive Language Model. *arXiv:2204.06745 [cs]*, April 2022. URL http://arxiv.org/abs/2204.06745.

Raghu Bollapragada, Richard Byrd, and Jorge Nocedal. Adaptive Sampling Strategies for Stochastic Optimization. *SIAM Journal on Optimization*, 28(4):3312–3343, January 2018a. ISSN 1052-6234. doi: 10.1137/17M1154679. URL https://epubs.siam.org/doi/abs/10.1137/17M1154679.

Raghu Bollapragada, Dheevatsa Mudigere, Jorge Nocedal, Hao-Jun Michael Shi, and Ping Tak Peter Tang. A Progressive Batching L-BFGS Method for Machine Learning. *arxiv:1802.05374[cs, math, stat]*, May 2018b. URL http://arxiv.org/abs/1802.05374.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, December 2020. URL http://arxiv.org/abs/2005.14165.

Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. Membership Inference Attacks From First Principles. *arxiv:2112.03570[cs]*, April 2022. doi: 10.48550/arXiv.2112.03570. URL http://arxiv.org/abs/2112.03570.

Ivan Chelombiev, Daniel Justus, Douglas Orr, Anastasia Dietrich, Frithjof Gressmann, Alexandros Koliousis, and Carlo Luschi. GroupBERT: Enhanced Transformer Architecture with Efficient Grouped Structures. *arxiv:2106.05822 [cs]*, June 2021. URL https://arxiv.org/abs/2106.05822v1.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. PaLM: Scaling Language Modeling with Pathways. *arXiv:2204.02311 [cs]*, April 2022. URL http://arxiv.org/abs/2204.02311.

Hyung Won Chung, Thibault Fevry, Henry Tsai, Melvin Johnson, and Sebastian Ruder. Rethinking Embedding Coupling in Pre-trained Language Models. In *International Conference on Learning Representations*, September 2020. URL https://openreview.net/forum?id=xpFFI_NtgpW.

Aidan Clark, Diego de las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, George van den Driessche, Eliza Rutherford, Tom Hennigan, Matthew Johnson, Katie Millican, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Jack Rae, Erich Elsen, Koray Kavukcuoglu, and Karen Simonyan. Unified Scaling Laws for Routed Language Models. *arXiv:2202.01169 [cs]*, February 2022. URL http://arxiv.org/abs/2202.01169.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *International Conference on Learning Representations*, September 2019. URL https://openreview.net/forum?id=r1xMH1BtvB.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Pre-Training Transformers as Energy-Based Cloze Models. *arXiv:2012.08561 [cs]*, December 2020. URL http://arxiv.org/abs/2012.08561.

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc Le. Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing. In *Advances in Neural Information Processing Systems*, volume 33, pp. 4271–4282. Curran Associates, Inc., 2020. URL https://papers.nips.cc/paper/2020/hash/2cd2915e69546904e4e5d4a2ac9e1652-Abstract.html.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *arxiv:2205.14135[cs]*, May 2022. doi: 10.48550/arXiv.2205.14135. URL http://arxiv.org/abs/2205.14135.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language Modeling with Gated Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 933–941. PMLR, July 2017. URL https://proceedings.mlr.press/v70/dauphin17a.html.

Boris Dayma, Suraj Patil, Pedro Cuenca, Khalid Saifullah, Tanishq Abraham, Phúc Lê Khác, Luke Melas, and Ritobrata Ghosh. DALL·E Mini, July 2021. URL https://github.com/borisdayma/dalle-mini.

Soham De, Abhay Yadav, David Jacobs, and Tom Goldstein. Big Batch SGD: Automated Inference using Adaptive Batch Sizes. *arxiv:1610.05792[cs, math, stat]*, April 2017. URL http://arxiv.org/abs/1610.05792.

Mostafa Dehghani, Yi Tay, Anurag Arnab, Lucas Beyer, and Ashish Vaswani. The Efficiency Misnomer. In *International Conference on Learning Representations*, September 2021. URL https://openreview.net/forum?id=iulEMLYh1uR.

Tim Dettmers. TPUs vs GPUs for Transformers (BERT), October 2018. URL https://timdettmers.com/2018/10/17/tpus-vs-gpus-for-transformers-bert/.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019. URL http://arxiv.org/abs/1810.04805.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929 [cs]*, June 2021. URL http://arxiv.org/abs/2010.11929.

Francesco Fusco, Damian Pascual, and Peter Staar. pNLP-Mixer: An Efficient all-MLP Architecture for Language. *arxiv:2202.04350 [cs]*, February 2022. URL https://arxiv.org/abs/2202.04350v1.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *arXiv:2101.00027 [cs]*, December 2020. URL http://arxiv.org/abs/2101.00027.

Jonas Geiping, Liam H. Fowl, W. Ronny Huang, Wojciech Czaja, Gavin Taylor, Michael Moeller, and Tom Goldstein. Witches' Brew: Industrial Scale Data Poisoning via Gradient Matching. In *International Conference on Learning Representations*, April 2021. URL https://openreview.net/forum?id=01olnfLIbD.

Shahriar Golchin, Mihai Surdeanu, Nazgol Tavabi, and Ata Kiapour. A Compact Pretraining Approach for Neural Language Models. *arxiv:2208.12367[cs]*, August 2022. doi: 10.48550/arXiv.2208.12367. URL http://arxiv.org/abs/2208.12367.

Zachary Gold and Mark Latonero. Robots Welcome: Ethical and Legal Considerations for Web Crawling and Scraping. *Washington Journal of Law, Technology & Arts*, 13(3):275–312, 2017. URL https://heinonline.org/HOL/P?h=hein.journals/washjolta13&i=283.

Xiaotao Gu, Liyuan Liu, Hongkun Yu, Jing Li, Chen Chen, and Jiawei Han. On the Transformer Growth for Progressive BERT Training. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5174–5180, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.406. URL https://aclanthology.org/2021.naacl-main.406.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing. *arXiv:2111.09543 [cs]*, December 2021. URL http://arxiv.org/abs/2111.09543.

Danny Hernandez, Tom Brown, Tom Conerly, Nova DasSarma, Dawn Drain, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Tom Henighan, Tristan Hume, Scott Johnston, Ben Mann, Chris Olah, Catherine Olsson, Dario Amodei, Nicholas Joseph, Jared Kaplan, and Sam McCandlish. Scaling Laws and Interpretability of Learning from Repeated Data. *arxiv:2205.10487[cs]*, May 2022. URL http://arxiv.org/abs/2205.10487.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training Compute-Optimal Large Language Models. *arXiv:2203.15556 [cs]*, March 2022. URL http://arxiv.org/abs/2203.15556.

Sara Hooker. The hardware lottery. *Communications of the ACM*, 64(12):58–65, November 2021. ISSN 0001-0782. doi: 10.1145/3467017. URL https://doi.org/10.1145/3467017.

Le Hou, Richard Yuanzhe Pang, Tianyi Zhou, Yuexin Wu, Xinying Song, Xiaodan Song, and Denny Zhou. Token Dropping for Efficient BERT Pretraining. *arXiv:2203.13240 [cs]*, March 2022. URL http://arxiv.org/abs/2203.13240.

Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. Transformer Quality in Linear Time. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 9099–9117. PMLR, June 2022. URL https://proceedings.mlr.press/v162/hua22a.html.

Like Hui and Mikhail Belkin. Evaluation of Neural Architectures Trained with Square Loss vs Cross-Entropy in Classification Tasks. *arXiv:2006.07322 [cs, stat]*, October 2021. URL http://arxiv.org/abs/2006.07322.

Forrest N. Iandola, Albert E. Shaw, Ravi Krishna, and Kurt W. Keutzer. SqueezeBERT: What can computer vision teach NLP about efficient neural networks? *arXiv:2006.11316 [cs]*, June 2020. URL http://arxiv.org/abs/2006.11316.

Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting Predictions from Training Data. *arxiv:2202.00622[cs, stat]*, February 2022. doi: 10.48550/arXiv.2202.00622. URL http://arxiv.org/abs/2202.00622.

Peter Izsak, Moshe Berchansky, and Omer Levy. How to Train BERT with an Academic Budget. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 10644–10652, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.831. URL https://aclanthology.org/2021.emnlp-main.831.

Mojan Javaheripi, Shital Shah, Subhabrata Mukherjee, Tomasz L. Religa, Caio C. T. Mendes, Gustavo H. de Rosa, Sebastien Bubeck, Farinaz Koushanfar, and Debadeepta Dey. LiteTransformerSearch: Training-free On-device Search for Efficient Autoregressive Language Models. *arXiv:2203.02094 [cs]*, March 2022. URL http://arxiv.org/abs/2203.02094.

Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic Generalization Measures and Where to Find Them. *arXiv:1912.02178 [cs, stat]*, December 2019. URL http://arxiv.org/abs/1912.02178.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for Natural Language Understanding. *arXiv:1909.10351 [cs]*, October 2020. URL http://arxiv.org/abs/1909.10351.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020. doi: 10.1162/tacl_a_00300. URL https://aclanthology.org/2020.tacl-1.5.

Prabhu Kaliamoorthi, Aditya Siddhant, Edward Li, and Melvin Johnson. Distilling Large Language Models into Tiny and Effective Students using pQRNN. *arxiv: 2101.08890 [cs]*, January 2021. URL https://arxiv.org/abs/2101.08890v1.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models. *arxiv:2001.08361[cs, stat]*, January 2020. doi: 10.48550/arXiv.2001.08361. URL http://arxiv.org/abs/2001.08361.

Guolin Ke, Di He, and Tie-Yan Liu. Rethinking Positional Encoding in Language Pre-training. In *International Conference on Learning Representations*, September 2020. URL https://openreview.net/forum?id=09-528y2Fgf.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, San Diego, May 2015. URL http://arxiv.org/abs/1412.6980.

Aran Komatsuzaki. One Epoch Is All You Need. *arXiv:1906.06669 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1906.06669.

Taku Kudo. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 66–75, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1007. URL https://aclanthology.org/P18-1007.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *EMNLP (Demonstration)*, July 2019. URL https://openreview.net/forum?id=S1EyQGf_bH.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *International Conference on Learning Representations*, September 2019. URL https://openreview.net/forum?id=H1eA7AEtvS.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating Training Data Makes Language Models Better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8424–8445, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.577. URL https://aclanthology.org/2022.acl-long.577.

James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. FNet: Mixing Tokens with Fourier Transforms. *arxiv:2105.03824 [cs]*, May 2021. URL https://arxiv.org/abs/2105.03824v3.

Tao Lei, Ran Tian, Jasmijn Bastings, and Ankur P. Parikh. Simple Recurrence Improves Masked Language Models. *arxiv:2205.11588[cs]*, May 2022. URL http://arxiv.org/abs/2205.11588.

Conglong Li, Minjia Zhang, and Yuxiong He. Curriculum Learning: A Regularization Method for Efficient and Stable Billion-Scale GPT Model Pre-Training. *arXiv:2108.06084 [cs]*, February 2022. URL http://arxiv.org/abs/2108.06084.

Frederick Liu, Siamak Shakeri, Hongkun Yu, and Jing Li. EncT5: Fine-tuning T5 Encoder for Non-autoregressive Tasks. *arxiv:2110.08426[cs]*, October 2021a. doi: 10.48550/arXiv.2110.08426. URL http://arxiv.org/abs/2110.08426.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the Variance of the Adaptive Learning Rate and Beyond. In *International Conference on Learning Representations*, March 2020a. URL https://openreview.net/forum?id=rkgz2aEKDr.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the Difficulty of Training Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5747–5763, Online, November 2020b. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.463. URL https://aclanthology.org/2020.emnlp-main.463.

Liyuan Liu, Jialu Liu, and Jiawei Han. Multi-head or Single-head? An Empirical Comparison for Transformer Training. *arxiv:2106.09650[cs]*, June 2021b. doi: 10.48550/arXiv.2106.09650. URL http://arxiv.org/abs/2106.09650.

Xiaoyu Liu, Jiahao Su, and Furong Huang. Tuformer: Data-driven Design of Transformers for Improved Generalization or Efficiency. In *International Conference on Learning Representations*, September 2021c. URL https://openreview.net/forum?id=V0A5g83gdQ_.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692 [cs]*, July 2019. URL http://arxiv.org/abs/1907.11692.

Zeyu Liu, Yizhong Wang, Jungo Kasai, Hannaneh Hajishirzi, and Noah A. Smith. Probing Across Time: What Does RoBERTa Know and When? In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 820–842, Punta Cana, Dominican Republic, November 2021d. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.71. URL https://aclanthology.org/2021.findings-emnlp.71.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *arXiv:1711.05101 [cs, math]*, November 2017. URL http://arxiv.org/abs/1711.05101.

Stephen Merity. Single Headed Attention RNN: Stop Thinking With Your Head. *arXiv:1911.11423 [cs]*, November 2019. URL http://arxiv.org/abs/1911.11423.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed Precision Training. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1gs9JgRZ.

Sören Mindermann, Jan Brauner, Muhammed Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N. Gomez, Adrien Morisot, Sebastian Farquhar, and Yarin Gal. Prioritized Training on Points that are Learnable, Worth Learning, and Not Yet Learnt. *arxiv:2206.07137[cs]*, June 2022. doi: 10.48550/arXiv.2206.07137. URL http://arxiv.org/abs/2206.07137.

Subhabrata Mukherjee, Ahmed Hassan Awadallah, and Jianfeng Gao. XtremeDistilTransformers: Task Transfer for Task-agnostic Distillation. *arXiv:2106.04563 [cs]*, June 2021. URL http://arxiv.org/abs/2106.04563.

Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://papers.nips.cc/paper/2019/hash/f1748d6b0fd9d439f71450117eba2725-Abstract.html.

Vaishnavh Nagarajan and J. Zico Kolter. Generalization in Deep Networks: The Role of Distance from Initialization. *arXiv:1901.01672 [cs, stat]*, January 2019. URL http://arxiv.org/abs/1901.01672.

Sharan Narang, Hyung Won Chung, Yi Tay, Liam 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. Do Transformer Modifications Transfer Across Implementations and Applications? In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5758–5773, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.465. URL https://aclanthology.org/2021.emnlp-main.465.

Shar Narasimhan. NVIDIA Clocks World's Fastest BERT Training Time and Largest Transformer Based Model, Paving Path For Advanced Conversational AI, August 2019. URL https://developer.nvidia.com/blog/training-bert-with-gpus/.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Łukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. Hierarchical Transformers Are More Efficient Language Models. *arxiv:2110.13711[cs]*, April 2022. URL http://arxiv.org/abs/2110.13711.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS 2017 Autodiff Workshop*, Long Beach, CA, 2017. URL https://openreview.net/forum?id=BJJsrmfCZ.

Bo Peng. RWKV-LM. Zenodo, August 2021. URL https://zenodo.org/record/5196577.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. *OpenAI*, pp. 24, 2019.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Ed Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. Scaling Language Models: Methods, Analysis & Insights from Training Gopher. *arXiv:2112.11446 [cs]*, January 2022. URL http://arxiv.org/abs/2112.11446.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv:1910.10683 [cs, stat]*, July 2020. URL http://arxiv.org/abs/1910.10683.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 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, pp. 3505–3506, New York, NY, USA, August 2020. Association for

Computing Machinery. ISBN 978-1-4503-7998-4. doi: 10.1145/3394486.3406703. URL https://doi.org/10.1145/3394486.3406703.

Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. {ZeRO-Offload}: Democratizing {Billion-Scale} Model Training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 551–564, 2021. ISBN 978-1-939133-23-6. URL https://www.usenix.org/conference/atc21/presentation/ren-jie.

Oliver Richter and Roger Wattenhofer. Normalized Attention Without Probability Cage. *arXiv:2005.09561 [cs, stat]*, May 2020. URL http://arxiv.org/abs/2005.09561.

Aurko Roy, Rohan Anil, Guangda Lai, Benjamin Lee, Jeffrey Zhao, Shuyuan Zhang, Shibo Wang, Ye Zhang, Shen Wu, Rigel Swavely, Tao, Yu, Phuong Dao, Christopher Fifty, Zhifeng Chen, and Yonghui Wu. N-Grammer: Augmenting Transformers with latent n-grams. *arxiv:2207.06366[cs]*, July 2022. doi: 10.48550/arXiv.2207.06366. URL http://arxiv.org/abs/2207.06366.

Christian Sarofeen, Piotr Bialecki, Jie Jiang, Kevin Stephano, Masaki Kozuki, Neal Vaidya, and Stas Bekman. Introducing nvFuser, a deep learning compiler for PyTorch, August 2022. URL https://pytorch.org/blog/introducing-nvfuser-a-deep-learning-compiler-for-pytorch/.

Teven Le Scao, Thomas Wang, Daniel Hesslow, Lucile Saulnier, Stas Bekman, M. Saiful Bari, Stella Biderman, Hady Elsahar, Jason Phang, Ofir Press, Colin Raffel, Victor Sanh, Sheng Shen, Lintang Sutawika, Jaesung Tae, Zheng Xin Yong, Julien Launay, and Iz Beltagy. What Language Model to Train if You Have One Million GPU Hours? In *Challenges* {\&, April 2022. URL https://openreview.net/forum?id=rI7BL3fHIZq.

Avi Schwarzschild. Easy-To-Hard, October 2021. URL https://github.com/aks2203/easy-to-hard.

Thibault Sellam, Steve Yadlowsky, Ian Tenney, Jason Wei, Naomi Saphra, Alexander D'Amour, Tal Linzen, Jasmijn Bastings, Iulia Raluca Turc, Jacob Eisenstein, Dipanjan Das, and Ellie Pavlick. The MultiBERTs: BERT Reproductions for Robustness Analysis. In *International Conference on Learning Representations*, March 2022. URL https://openreview.net/forum?id=K0E_F0gFDgA.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive Learning Rates with Sublinear Memory Cost. *arxiv:1804.04235[cs, stat]*, April 2018. doi: 10.48550/arXiv.1804.04235. URL http://arxiv.org/abs/1804.04235.

Sheng Shen, Pete Walsh, Kurt Keutzer, Jesse Dodge, Matthew Peters, and Iz Beltagy. Staged Training for Transformer Language Models. *arXiv:2203.06211 [cs]*, March 2022. URL http://arxiv.org/abs/2203.06211.

Sam Shleifer, Jason Weston, and Myle Ott. NormFormer: Improved Transformer Pretraining with Extra Normalization. *arXiv:2110.09456 [cs]*, November 2021. URL http://arxiv.org/abs/2110.09456.

Leslie N. Smith and Nicholay Topin. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates. *arXiv:1708.07120 [cs, stat]*, May 2018. URL http://arxiv.org/abs/1708.07120.

David So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V. Le. Searching for Efficient Transformers for Language Modeling. In *Advances in Neural Information Processing Systems*, May 2021. URL https://openreview.net/forum?id=bzpkxS_JVsI.

Sharath Nittur Sridhar, Anthony Sarah, and Sairam Sundaresan. TrimBERT: Tailoring BERT for Trade-offs. *arXiv:2202.12411 [cs]*, February 2022. URL http://arxiv.org/abs/2202.12411.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. *arxiv:2104.09864 [cs]*, April 2021. URL https://arxiv.org/abs/2104.09864v2.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive Attention Span in Transformers. *arXiv:1905.07799 [cs, stat]*, August 2019. URL http://arxiv.org/abs/1905.07799.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient Knowledge Distillation for BERT Model Compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4323–4332, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1441. URL https://aclanthology.org/D19-1441.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. MobileBERT: A Compact Task-Agnostic BERT for Resource-Limited Devices. *arXiv:2004.02984 [cs]*, April 2020. URL http://arxiv.org/abs/2004.02984.

Richard Sutton. The Bitter Lesson. *Incomplete Ideas (blog)*, pp. 1, March 2019. URL http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

Liling Tan. What the bookcorpus?, December 2019. URL https://gist.github.com/alvations/4d2278e5a5fbcf2e07f49315c4ec1110.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long Range Arena: A Benchmark for Efficient Transformers. *arXiv:2011.04006 [cs]*, November 2020a. URL http://arxiv.org/abs/2011.04006.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient Transformers: A Survey. *arXiv:2009.06732 [cs]*, September 2020b. URL http://arxiv.org/abs/2009.06732.

Yi Tay, Mostafa Dehghani, Jinfeng Rao, William Fedus, Samira Abnar, Hyung Won Chung, Sharan Narang, Dani Yogatama, Ashish Vaswani, and Donald Metzler. Scale Efficiently: Insights from Pretraining and Finetuning Transformers. In *International Conference on Learning Representations*, September 2021. URL https://openreview.net/forum?id=f2OYVDyfIB.

Yi Tay, Mostafa Dehghani, Samira Abnar, Hyung Won Chung, William Fedus, Jinfeng Rao, Sharan Narang, Vinh Q. Tran, Dani Yogatama, and Donald Metzler. Scaling Laws vs Model Architectures: How does Inductive Bias Influence Scaling? *arxiv:2207.10551[cs]*, July 2022a. doi: 10.48550/arXiv.2207.10551. URL http://arxiv.org/abs/2207.10551.

Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Xavier Garcia, Dara Bahri, Tal Schuster, Huaixiu Steven Zheng, Neil Houlsby, and Donald Metzler. Unifying Language Learning Paradigms. *arxiv:2205.05131[cs]*, May 2022b. URL http://arxiv.org/abs/2205.05131.

Marcos Treviso, Tianchu Ji, Ji-Ung Lee, Betty van Aken, Qingqing Cao, Manuel R. Ciosici, Michael Hassid, Kenneth Heafield, Sara Hooker, Pedro H. Martins, André F. T. Martins, Peter Milder, Colin Raffel, Edwin Simpson, Noam Slonim, Niranjan Balasubramanian, Leon Derczynski, and Roy Schwartz. Efficient Methods for Natural Language Processing: A Survey. *arxiv:2209.00099[cs]*, August 2022. URL http://arxiv.org/abs/2209.00099.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-Read Students Learn Better: On the Importance of Pre-training Compact Models. *arXiv:1908.08962 [cs]*, September 2019. URL http://arxiv.org/abs/1908.08962.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. URL http://arxiv.org/abs/1706.03762.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *International Conference on Learning Representations*, September 2018. URL https://openreview.net/forum?id=rJ4km2R5t7.

Shibo Wang and Pankaj Kanwar. BFloat16: The secret to high performance on Cloud TPUs, August 2019. URL https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus/.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-Attention with Linear Complexity. *arxiv:2006.04768v3 [cs]*, June 2020a. URL https://arxiv.org/abs/2006.04768v3.

Thomas Wang, Adam Roberts, Daniel Hesslow, Teven Le Scao, Hyung Won Chung, Iz Beltagy, Julien Launay, and Colin Raffel. What Language Model Architecture and Pretraining Objective Work Best for Zero-Shot Generalization? *arXiv:2204.05832 [cs, stat]*, April 2022. URL http://arxiv.org/abs/2204.05832.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. *arXiv:2002.10957 [cs]*, April 2020b. URL http://arxiv.org/abs/2002.10957.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural Network Acceptability Judgments. *arxiv:1805.12471[cs]*, October 2019. doi: 10.48550/arXiv.1805.12471. URL http://arxiv.org/abs/1805.12471.

Alexander Wettig, Tianyu Gao, Zexuan Zhong, and Danqi Chen. Should You Mask 15% in Masked Language Modeling? *arxiv: 2202.08005 [cs]*, February 2022. URL https://arxiv.org/abs/2202.08005v1.

Noam Wies, Yoav Levine, Daniel Jannai, and Amnon Shashua. Which transformer architecture fits my data? A vocabulary bottleneck in self-attention. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 11170–11181. PMLR, July 2021. URL https://proceedings.mlr.press/v139/wies21a.html.

Rachel Wilka, Rachel Landy, and Scott A. McKinney. How Machines Learn: Where Do Companies Get Data for Machine Learning and What Licenses Do They Need. *Washington Journal of Law, Technology & Arts*, 13(3):217–244, 2017. URL https://heinonline.org/HOL/P?h=hein.journals/washjolta13&i=226.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *arXiv:1910.03771 [cs]*, July 2020. URL http://arxiv.org/abs/1910.03771.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arxiv:1609.08144[cs]*, October 2016. URL http://arxiv.org/abs/1609.08144.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On Layer Normalization in the Transformer Architecture. *arXiv:2002.04745 [cs, stat]*, June 2020. URL http://arxiv.org/abs/2002.04745.

Abhay Yadav. Making L-BFGS Work with Industrial-Strength Nets. In *BMVC 2020*, pp. 13, 2020.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv:1906.08237 [cs]*, January 2020. URL http://arxiv.org/abs/1906.08237.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes. In *International Conference on Learning Representations*, September 2019. URL https://openreview.net/forum?id=Syx4wnEtvH.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for Longer Sequences. *arXiv:2007.14062 [cs, stat]*, January 2021. URL http://arxiv.org/abs/2007.14062.

Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. *arXiv:1910.07467 [cs, stat]*, October 2019. URL http://arxiv.org/abs/1910.07467.

Minjia Zhang and Yuxiong He. Accelerating Training of Transformer-Based Language Models with Progressive Layer Dropping. In *Advances in Neural Information Processing Systems*, volume 33, pp. 14011–14023, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/a1140a3d0df1c81e24ae954d935e8926-Abstract.html?fbclid=IwAR2bOnieMB2yBSK6Ks9_RVAkPHfUnTQLliZzJEzp89cLiIvErF_zf6efu6c.

Chen Zhu, Renkun Ni, Zheng Xu, Kezhi Kong, W. Ronny Huang, and Tom Goldstein. GradInit: Learning to Initialize Neural Networks for Stable and Efficient Training. *arxiv:2102.08098[cs]*, November 2021. doi: 10.48550/arXiv.2102.08098. URL http://arxiv.org/abs/2102.08098.

Appendix

A Other Modifications

A few recent developments not included in this study are Roy et al. (2022), Shen et al. (2022), and Mindermann et al. (2022). Modifications further not included in this study are more involved initialization (Zhu et al., 2021), additional objective modifications (Müller et al., 2019), progressive growth (Gu et al., 2021; Shen et al., 2022), convolutional variants (Iandola et al., 2020; Chelombiev et al., 2021; So et al., 2021), sequence recurrence (Lei et al., 2022) and TUPE embeddings (Ke et al., 2020).

B Additional Information

Additional results concerning architecture modifications can be found in Table 8 and Table 9. Additional results for training modifications can be found in Table 10. Not all results remarked on in the main body (especially for variations that did not work, marked in gray) are accompanied by raw results in this appendix, but can be computed using the provided implementation.

B.1 References for Table 1

The maximal floating point operations referenced in Table 1 are based on the following published numbers. For TPU specs, according to https://cloud.google.com/tpu/docs/system-architecture-tpu-vm we find 275 TFLOP/s in bfloat16 precision for the TPUv4 and 123 TFLOP/s for the TPUv3, each per chip. The V100 peak performance is given as 125 TFLOP/s in https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf in ”TFLOPS of mixed precision”. Some NVIDIA datasheets also reference TFLOP/s with sparsity, which are not applicable in the context of this work. The Titan RTX comes out at 130.5 TFLOP/s, in ”Peak FP16 Tensor TFLOPS with FP32 Accumulate” as described in https://images.nvidia.com/aem-dam/en-zz/Solutions/geforce/ampere/pdf/NVIDIA-ampere-GA102-GPU-Architecture-Whitepaper-V1.pdf. For the A6000, we find 154.8 ”Peak BF16 Tensor TFLOPS with FP32

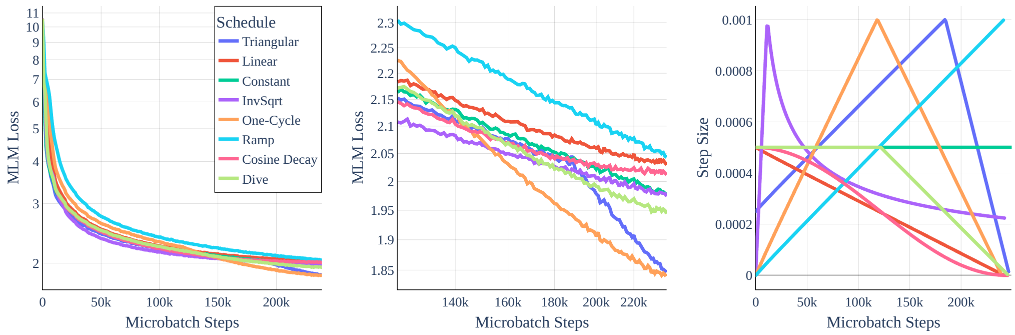

Figure 4: Extended version of Figure 2, including additional learning rate schedules.

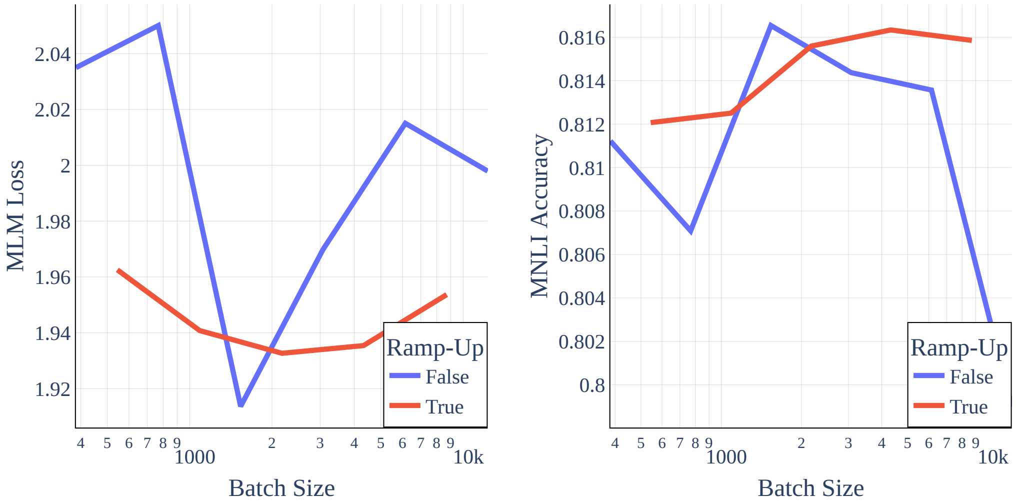

Figure 5: Partial variations of batch sizes with and without linear ramp-up. All experiments run with the training setup described in Section 4.3 for a day on a single GPU with mixed precision. Batch size is 4036 and dataset is `bookcorpus-wikipedia`. Downstream evaluation as described in Section 5. All values for pretraining on an A4000. Note the discrepancy between optimal pretraining batch size and optimal batch size for evaluation on MNLI when ramp-up is used, but also note that differences are overall barely significant.

	MNLI	SST-2	STSB	RTE	QNLI	QQP	MRPC	CoLA	GLUE
				Trained for 1 day on a **2080ti**					
crammed BERT	82.8 / 83.4	91.5	83.1	54.0	89.0	87.2	86.2	47.2	78.3
+with original data	81.7 / 82.0	91.3	82.3	51.8	88.7	86.9	85.5	**48.1**	77.6
+with original train	50.4 / 50.7	81.1	12.2	50.9	58.2	66.2	73.8	8.9	50.3
+with original arch.	58.7 / 57.8	79.8	16.6	50.9	55.4	71.1	70.1	7.3	52.0
+with minimal train mod.	80.2 / 80.5	89.6	82.7	**55.4**	86.6	86.4	84.1	39.0	76.0
+with minimal arch. mod.	81.7 / 82.5	91.2	79.2	54.5	87.7	86.4	83.0	38.8	76.1
				Trained for 1 day on an **A4000**					
crammed BERT	83.0 / 83.2	91.6	84.8	54.7	88.5	86.9	86.4	43.7	78.1
+with original data	81.5 / 81.8	91.0	81.8	49.5	88.3	86.8	84.5	43.2	76.5
+with original train	50.2 / 50.8	80.8	12.8	49.8	59.0	66.3	73.7	7.7	50.1
+with original arch.	58.0 / 56.5	79.4	17.0	51.6	54.2	70.6	74.1	8.2	52.2
+with minimal train mod.	80.0 / 80.4	89.3	84.2	55.2	86.5	86.4	86.3	40.1	76.5
+with minimal arch. mod.	82.1 / 82.6	91.5	79.9	54.7	87.9	86.6	82.9	35.4	76.0
				Trained for 1 day on an **A6000**					
crammed BERT	**83.9 / 84.1**	**92.2**	84.6	53.8	**89.5**	**87.3**	87.5	44.5	**78.6**
+ original data	82.2 / 82.7	92.0	83.6	49.8	**89.5**	87.0	85.9	42.5	77.3
+ original train	50.0 / 50.4	80.7	13.7	52.0	59.8	65.1	73.2	7.2	50.2
+ original arch.	35.4 / 35.2	49.1	-	52.7	49.5	0.0	0.0	0.0	27.7
+ minimal train mod.	81.9 / 82.6	91.4	**85.5**	54.9	88.2	87.0	**88.4**	43.6	78.1
+ minimal arch. mod.	83.2 / 83.5	91.7	82.0	52.0	88.9	86.8	83.6	38.3	76.7

Table 7: Extension of Table 5, including results on the other GPU types.

Accumulate” also in https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf. A4000 performance is actually less clear from this whitepaper, and estimated to be 88.45 TFLOP/s, based on it containing 192 tensor cores, compared to 336 for the A6000. For the RTX2080ti, the whitepaper at https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf reports 53.8 ”peak FP16 Tensor TFLOPS with FP32 Accumulate” for the reference edition. All total exaFLOP numbers are then computed based on these TFLOP/s numbers over the training time period described in each work.

Name	MLM Loss	MNLI-m	MNLI-mm	Tokens/Second
Modified Transformer	1.89	81.02	81.35	50946
DeepNarrow (12 Layers)	1.94	80.90	80.97	78396
DeepNarrow (24 Layers)	1.98	80.78	81.14	41289
$E = 128$	2.14	76.68	77.62	53267
FFN every 2 blocks	1.93	80.43	80.97	64774
FFN every 3 blocks	1.97	80.44	80.93	71634
FFN every 4 blocks	2.00	80.03	79.67	73319
$H = 512$	1.93	80.61	80.93	83718
$H = 1024$	1.95	80.07	80.68	32004
4 Layers	2.00	78.45	79.00	137127
6 Layers	1.93	79.49	79.82	96156
8 Layers	1.89	81.11	81.08	74248
10 Layers	1.89	81.02	81.21	61431
16 Layers	1.92	81.39	82.10	39406
24 Layers	2.01	80.64	80.97	26927
Recurrent (1-12)	2.40	77.46	77.81	52405
Recurrent (2-6)	2.04	80.45	80.73	53148
Recurrent (3-4)	2.00	80.78	81.33	51634
Recurrent (4-3)	1.98	80.95	81.26	51952
BERT-tiny	3.30	56.71	57.21	914694
BERT-mini	2.49	72.22	73.21	429593
BERT-Large (Izsak variant)	2.38	76.93	77.47	13448
Original BERT	7.54	35.45	35.22	41978
With decoder bias	1.89	80.97	81.20	51155
With $\varepsilon = 10^{-6}$ in Layer Norm	1.90	80.49	81.35	51728
Learned Embedding	1.88	80.51	81.03	52601
No Norm after Embedding	1.94	79.65	80.34	52175
No Final Norm	1.89	80.40	80.89	51207
No Skip of Head Transform	1.88	80.49	81.19	51728
No Rotational Embedding	1.88	80.91	81.52	53526
Post-LN	7.54	31.82	31.82	52270
With QKV bias	1.89	80.70	80.88	51112
With bias in Linear Layers	1.89	80.64	81.49	50584
12 Heads	1.88	81.75	81.99	47967

Table 8: Additional raw results for experiments considered in the main body. This table contains architecture variants for a prelimary architecture setup which contained 4 heads in the attention block, 12 layers and included rotary embeddings. First two blocks: Architectural variants as discussed in Section 4.2 (but for this preliminary variant). Third block: Ablation study of this model. All experiments run with the training setup described in Section 4.3 for a day on a single GPU with mixed precision. Batch size is 4032 and dataset is `bookcorpus-wikipedia`. Downstream evaluation as described in Section 5. All values for pretraining on an A4000.

Name	MLM Loss	MNLI	MNLI-mm	Tokens/Second
Modified Transformer	1.84	81.79	82.14	46431
DeepNarrow (12 Layers)	1.91	80.97	81.30	99717
DeepNarrow (24 Layers)	1.91	81.39	81.61	52558
$E = 128$	2.04	-	-	48468
FFN every 2 blocks	1.84	81.40	81.65	62134
FFN every 3 blocks	1.87	80.90	81.53	70685
FFN every 4 blocks	1.88	81.10	81.42	75163
$H = 512$	1.87	81.34	82.20	79116
$H = 1024$	1.94	80.63	80.97	28511
4 Layers	1.94	79.13	79.51	161034
6 Layers	1.87	80.48	80.84	115037
8 Layers	1.84	81.22	81.62	88652
10 Layers	1.82	81.25	82.31	71414
12 Layers	1.85	81.68	82.18	59346
18 Layers	1.90	81.02	81.82	40577
24 Layers	1.97	80.81	81.26	30455
Recurrent (1-12)	2.13	79.23	79.78	62318
Recurrent (2-6)	2.00	80.86	81.24	62677
Recurrent (3-4)	1.94	80.95	81.48	61772
Recurrent (4-3)	1.91	81.43	81.84	61596
BERT-Tiny Variant	3.51	56.10	56.60	1018443
BERT-Mini Variant	2.46	72.30	73.47	523061
BERT-Large Variant	2.12	79.50	79.84	17688
BERT-Large (Izsak variant)	2.37	76.81	77.56	13522
Original BERT	7.53	35.45	35.22	41362
With decoder bias	1.84	81.71	81.91	45996
With $\varepsilon = 10^{-6}$ in Layer Norm	1.83	81.55	82.13	45841
Learned Embedding	1.83	81.31	81.79	46608
No Norm after Embedding	1.89	81.38	81.15	46267
No Final Norm	1.85	80.67	80.87	46598
No Skip of Head Transform	1.83	82.03	82.19	46324
With QKV Bias	1.83	81.89	82.28	46469
With bias in Linear Layers	1.84	81.88	82.16	45629
4 Heads	1.88	81.22	81.77	40551
With Rotary Embedding	1.86	81.16	81.94	42257
Post-LN	7.54	35.21	35.17	46324
Fourier Attention	2.65	68.97	69.06	46634
GELU	1.832477	81.94	82.17	47779

Table 9: Additional raw results for experiments considered in the main body for the final architecture variant. First two blocks: Architectural variants as discussed in Section 4.2. Third block: Ablation study of finally adopted model. All experiments run with the training setup described in Section 4.3 for a day on a single GPU with mixed precision. Batch size is 4096 and dataset is `bookcorpus-wikipedia`. Downstream evaluation as described in Section 5. All values for pretraining on an A4000.

Name	MLM	MNLi-m	MNLI-mm	Tokens/Second
Original training recipe	7.28	60.65	60.31	49264
With Izsak Training recipe	2.06	79.90	80.30	46869
Minimal Modifications	2.03	78.78	79.36	47346
+Larger LR	1.99	80.25	80.50	46524
+One Cycle, +Larger LR	1.84	82.12	82.55	46843
+One Cycle, +Larger LR, +Clipping	1.84	81.79	82.14	46303
Sequence Curriculum (10%,20%,30%,50%,75%)	3.02	70.06	70.77	29359
Sequence Curriculum (+unfolding)	1.87	80.13	80.04	46014
Sequence Curriculum (20%,35%,50%,65%,85%)	1.90	79.86	79.80	45804
Adafactor	1.86	81.36	82.22	45997
Adam (classic WD formulation)	7.44	32.28	32.39	49598
SGD	7.46	59.30	58.02	47678
RADAM	7.50	32.74	32.95	48812
With Dropout activated	1.97	80.95	80.98	45198
With MLM masking 20%	2.06	80.76	81.48	45944
With MLM masking 40%	2.70	81.11	81.30	43467
With MLM masking 60%	3.41	80.62	80.88	40756

Table 10: Additional raw results for experiments considered in the main body for the final training variant, not otherwise mentioned. Batch size is 4096 and dataset is `bookcorpus-wikipedia`. Downstream evaluation as described in Section 5. All values for pretraining on an A4000.