Train Large, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers

Zhuohan Li [*1] **Eric Wallace** [*1] **Sheng Shen** [*1] **Kevin Lin** [*1]
Kurt Keutzer [1] **Dan Klein** [1] **Joseph E. Gonzalez** [1]

Abstract

Since hardware resources are limited, the objective of training deep learning models is typically to maximize accuracy subject to the time and memory constraints of training and inference. We study the impact of model size in this setting, focusing on Transformer models for NLP tasks that are limited by compute: self-supervised pretraining and high-resource machine translation. We first show that even though smaller Transformer models execute faster per iteration, wider and deeper models converge in significantly fewer steps. Moreover, this acceleration in convergence typically outpaces the additional computational overhead of using larger models. Therefore, the most compute-efficient training strategy is to counterintuitively train extremely large models but stop after a small number of iterations. This leads to an apparent trade-off between the training efficiency of large Transformer models and the inference efficiency of small Transformer models. However, we show that large models are more robust to compression techniques such as quantization and pruning than small models. Consequently, one can get the best of both worlds: heavily compressed, large models achieve higher accuracy than lightly compressed, small models.

1. Introduction

In the current deep learning paradigm, using more compute (e.g., increasing model size, dataset size, or training steps) typically leads to higher model accuracy (Brock et al., 2019; Raffel et al., 2019). This phenomenon is exacerbated by the recent success of self-supervised pretraining (Devlin et al., 2019; Hénaff et al., 2019), which allows training

[*]Equal contribution [1]UC Berkeley. Correspondence to: Zhuohan Li <zhuohan@cs.berkeley.edu>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

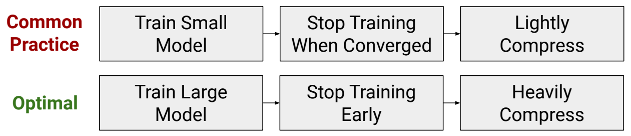
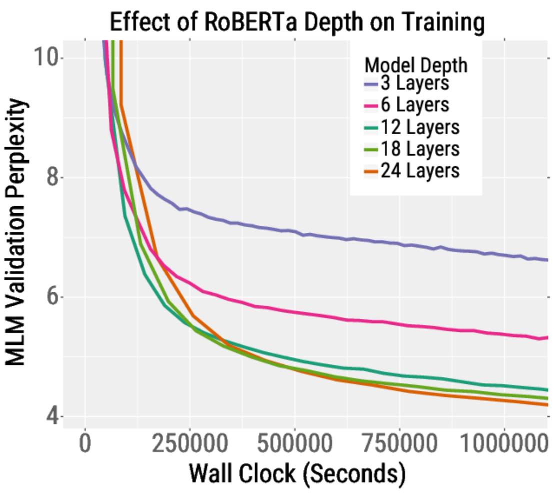


Figure 1. Under the usual presumption that models are trained to convergence, only small models that are fast-to-execute are feasible in resource-constrained settings. Our work shows that the most compute-efficient training scheme is instead to train very large models, stop them well short of convergence, and then heavily compress them to meet test-time constraints.

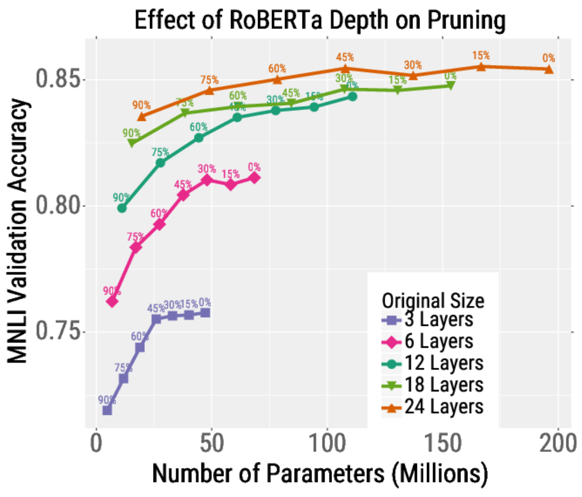
to scale to massive amounts of unlabeled data and very large neural models. Consequently, computational resources are increasingly the critical constraint on improving model accuracy. This constraint causes the (often implicit) goal of model training to be maximizing *compute efficiency*: how to achieve the highest model accuracy given a fixed amount of hardware and training time.

Maximizing compute efficiency requires rethinking common assumptions about model training. In particular, there is typically an implicit assumption that models must be trained *until convergence*, which makes larger models appear less viable for limited compute budgets. We challenge this assumption by demonstrating the opportunity to increase model size at the cost of convergence. Concretely, we show that the fastest way to train Transformer models (Vaswani et al., 2017) is to substantially *increase* model size but stop training very early.

In our experiments, we vary the width and depth of Transformer models and evaluate their training time and accuracy on self-supervised pretraining (RoBERTa (Liu et al., 2019b) trained on Wikipedia and BookCorpus) and machine translation (WMT14 English→French). For these tasks, we first show that larger models converge to lower validation error in fewer gradient updates than smaller models (Section 3). Moreover, this increase in convergence outpaces the additional computational overhead of using larger models—the most compute-efficient models are extremely large and stopped well short of convergence (e.g., Figure 2, left). We also show that this acceleration in wall-clock convergence



(a)



(b)

Figure 2. Increasing Transformer model size results in lower validation error as a function of *wall-clock time* and better test-time accuracy for a given *inference budget*. **(a)** demonstrates the training speedup for RoBERTa models of different sizes on the masked language modeling pretraining task. In **(b)**, we take RoBERTa checkpoints that have been pretrained for the *same* amount of wall-clock time and finetune them on a downstream dataset (MNLI). We then iteratively prune model weights to zero and find that the best models for a given test-time memory budget are ones which are trained large and then heavily compressed.

is largely a function of parameter count and only weakly influenced by model width, depth, and batch size.

Although larger models train faster, they also increase the computational and memory requirements of inference. This increased cost is especially problematic in real-world applications, where the cost of inference dominates the cost of training (Jouppi et al., 2017; Crankshaw et al., 2017; Metz, 2017). However, we show that for RoBERTa, this apparent trade-off can be reconciled with compression: large models are considerably more robust to compression as compared to small models (Section 4). Thus, large, heavily compressed models outperform small, lightly compressed models using comparable inference costs (e.g., Figure 2, right).

We finally analyze *when* and *why* large models train fast and compress well (Section 5). We show that the optimal model size is closely linked to the dataset size. In particular, large models perform favorably in big data settings where overfitting is a limited concern. We then analyze why larger models are more compressible by measuring the difference in weights when using quantized or sparse weight matrices. This error decreases as model size increases, i.e., greater overparameterization leads to easy-to-compress weights.

2. Experimental Setup

2.1. Tasks, Models, and Datasets

We train state-of-the-art models for two NLP tasks: self-supervised pretraining using masked language modeling and high-resource machine translation. We chose these tasks because accuracy continues to improve as models are made larger (Shazeer et al., 2018), trained for more steps (Liu et al., 2019b), and trained using larger batches (Raffel et al., 2019). Thus, a critical factor in improving accuracy for these tasks is to maximize the compute efficiency of training.

Self-supervised Pretraining (MLM) We closely follow the pretraining setup and model from RoBERTa (Liu et al., 2019b) with a few minor exceptions. We move the model's layer normalization layers (Ba et al., 2016) to the input of every sub-layer (often called *pre-norm*). This slightly improves results and stabilizes training (Wang et al., 2019b). We also use an input sequence length of 128 and a batch size of 8192, unless otherwise noted. For RoBERTa, we vary the depth in $\{3, 6, 12, 18, 24\}$, and the hidden size in $\{256, 512, 768, 1024, 1536\}$.

The dataset for pretraining RoBERTa is not publicly available. We instead follow BERT (Devlin et al., 2019) and concatenate the BookCorpus (Zhu et al., 2015) and a Wikipedia dump to use for training. Since the BookCorpus is no longer publicly available, we follow Devlin et al. (2019) and crawl http://smashwords.com. Our final dataset is roughly 3.4 billion words in total. We hold out a random 0.5% of the data for validation and report the masked language modeling (MLM) perplexity on this data. We also evaluate the model by finetuning on MNLI (Williams et al., 2018) and SST-2 (Socher et al., 2013). We found the variance in accuracy for these two tasks to be lower than the other GLUE tasks (Wang et al., 2019a).

Machine Translation For machine translation (MT) we train the standard Transformer architecture and hyperparameters on the WMT14 English$\rightarrow$French dataset. We use the standard dataset splits: 36M sentences for training, newstest2013 for validation, and newstest2014 for testing. We follow standard practice and report tokenized case-sensitive BLEU (Papineni et al., 2002) with compound splitting (Vaswani et al., 2017). We vary the model depth in $\{2, 6, 8\}$ and hidden size in $\{128, 256, 512, 1024, 2048\}$.

2.2. Evaluation Metrics: FLOPs and Wall-Clock Time

Recent work on resource-constrained training uses the total number of training steps (Li et al., 2020) or the total number of training FLOPs (Schwartz et al., 2019; Clark et al., 2020) as the main evaluation metric. These metrics do not adequately capture the true training time. In particular, reporting gradient steps does not account for the cost of using bigger batches or models. Moreover, although reporting FLOPs is useful for comparison as it is hardware-agnostic, it neglects the fact that parallel operations are significantly cheaper than sequential operations on modern hardware.

We instead directly report wall-clock time as our main evaluation metric.[1] Since the runtime varies across machines (the hardware setups are different, the jobs are not isolated, etc.), we use a single machine to benchmark the time per gradient step for each model size. In particular, we train models and wait for the time per gradient step to stabilize, and then we use the average time over 100 steps to calculate the training duration. We conduct the timing on one NVIDIA 16GB V100 GPU and use gradient accumulation to fit larger models and batches. In order to be fair to smaller models, we increase the batch size to the largest size that fits in memory. This means that smaller models use fewer gradient accumulation steps and thus take less time per gradient step (which we confirmed empirically). We use Tensor2Tensor (Vaswani et al., 2018) for MT and fairseq (Ott et al., 2019) for RoBERTa. We train using a mix of v3-8 TPUs and 8xV100 GPUs for both tasks.

3. Larger Models Train Faster

Wider and deeper Transformer models are more sample-efficient than small models: they reach the same level of performance using fewer gradient steps (Figures 3–5). Moreover, this increase in convergence outpaces the additional computational overhead from increasing model size, even though we need to use more steps of gradient accumulation. Consequently, after adjusting for wall-clock time, the larger models are *faster* to train than smaller models (Figures 4–5).

[1]We also report selected learning curves as a function of FLOPs in Appendix A.1. These curves show that our conclusion that larger models are faster to train is not specific to our hardware setup.

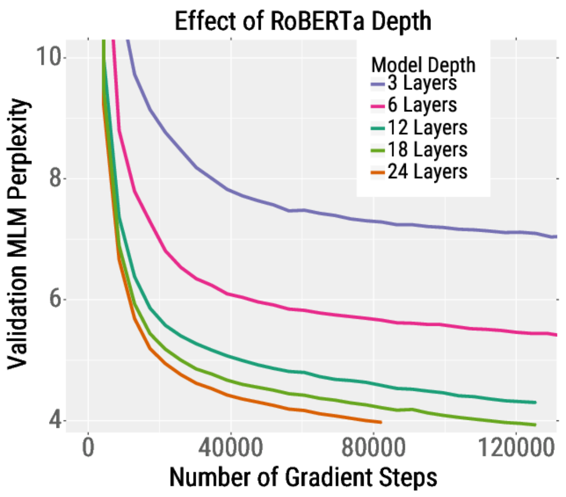


Figure 3. Deeper RoBERTa models converge faster than shallow models with respect to the gradient steps (wall-clock time shown in Figure 2, left).

Increase Model Width and Sometimes Depth For the masked language modeling task, the validation perplexity weakly depends on the shape of the model. Instead, the total number of model parameters is the key determiner of the convergence rate. Thus, increasing either the width or the depth is effective at accelerating model training. On the other hand, the preferred way to scale models for MT is to increase their width as wider models usually outperform deep models in final performance (Vaswani et al., 2017; Shazeer et al., 2018).

Increase Model Size, Not Batch Size Another factor that affects the training efficiency is the batch size. In particular, there is a trade-off between using fast-to-execute small batches and slow-but-accurate large batches. We study the effect of scaling batch size because it provides an alternative to scaling model size. In particular, *what if we use gradient accumulation to increase the batch size rather than the model size?* We vary the batch size for the 12 layer, 768H model and increase the learning rate as is common practice (Goyal et al., 2017; Liu et al., 2019b). We report the best found learning rate values in Table 1 in Appendix A.

We show the training curves in Figure 13 in Appendix A. Bigger batch sizes cause the model to converge in fewer steps. However, when adjusting for wall-clock time, increasing the batch size beyond a certain point only provides marginal improvements.[2] In particular, varying the batch size has little impact when training with a batch size in the

[2]Note that our timing is done by accumulating gradients on a single GPU machine. For multi-GPU setups, the cost of accumulating gradients is lower as it naturally helps to balance out uneven runtimes across workers (Ott et al., 2018). In this setup, the wall-clock improvements from increasing batch sizes by accumulating gradients may be slightly larger.

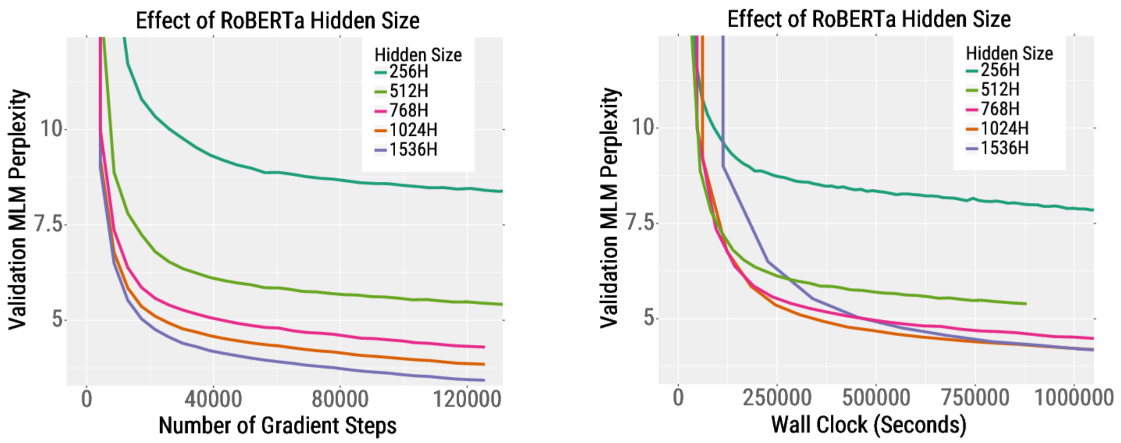


Figure 4. Wider models converge faster than narrower models as function of both gradient steps (left plot) and wall-clock time (right plot).

range from 2048–16384. This aligns with the findings of McCandlish et al. (2018): training efficiency is maximized when models are trained near some *critical batch size*.

An additional downside of increasing the batch size is that it requires simultaneously tuning the learning rate. On the other hand, scaling model size provides improvements in training efficiency *without* adjusting any hyperparameters. Overall, our results show that one should increase the batch size (and learning rate) until the critical batch size region is reached and then to focus on increasing model size.

Larger Models Are Not Harder to Finetune Although the larger models minimize validation MLM perplexity faster, one concern is that they may not minimize downstream task error faster. For instance, larger models may overfit on small downstream datasets. We investigate this by training RoBERTa models of different sizes and stopping them when they reach the same MLM perplexity (the larger models have been trained for *less* wall-clock time). We then finetune each model using the RoBERTa finetuning hyperparameters (Liu et al., 2019b) on MNLI and SST-2. We report the model accuracies in Table 2 in Appendix B. All models reach comparable accuracies (in fact, the larger models typically outperform the smaller ones), which shows that larger models are not more difficult to finetune.

Returns Diminish As Size Increases For both RoBERTa and MT, the largest models have reached the point where they stop improving convergence with respect to wall-clock time. For example, the largest model for MT (6L, 2048H) starts to converge slower with respect to wall-clock time than the second-largest model (6L, 1024H). These diminishing returns occur because (1) the per-step convergence improvements from using larger models decreases as the model gets larger and (2) the computational overhead increases as our hardware becomes increasingly compute-bound. We further analyze when and why returns diminish in Section 5.

4. Larger Models Compress Better

Although the most compute-efficient *training* scheme is to use larger models, this results in models which are less *inference* efficient. Here, we demonstrate how to get the best of both worlds. In particular, we show that since large models are more compressible than small models, they can outperform small models while using similar inference costs.

4.1. Compression Methodology and Evaluation

Compression Methods Model compression methods reduce the inference costs of trained models. For example, model compression can reduce inference latency to enable real-time applications like simultaneous MT (See et al., 2016) or reduce memory usage to save energy for mobile devices (Han et al., 2016). We focus on compression methods which are *fast* to perform—methods which require significant amounts of compute will negate the speedup from using larger models.[3] In particular, we consider two compression techniques: quantization (Section 4.2) and pruning (Section 4.3), as well as their combination.[4] Quantization stores model weights in low precision formats to (1) accelerate operations when using hardware with reduced precision support and (2) reduce overall memory footprint (Han et al., 2016; Dong et al., 2019). Pruning sets neural network weights to zero to (1) remove operations and (2) reduce the memory footprint when models are stored in sparse matrix formats (LeCun et al., 1990; Han et al., 2015). We apply both quantization and pruning post-hoc to the finetuned models to limit the additional computational overhead.

[3]For example, we avoid using model distillation methods because they can add a significant computational overhead (Sanh et al., 2019; Turc et al., 2019) or cause a significant degradation in accuracy (Liu et al., 2019a; Sun et al., 2019).

[4]We also experiment with *parameter sharing* (Lan et al., 2020; Dehghani et al., 2019)—tying the weights of the Transformer layers together—and find that it slows convergence (see Appendix C).

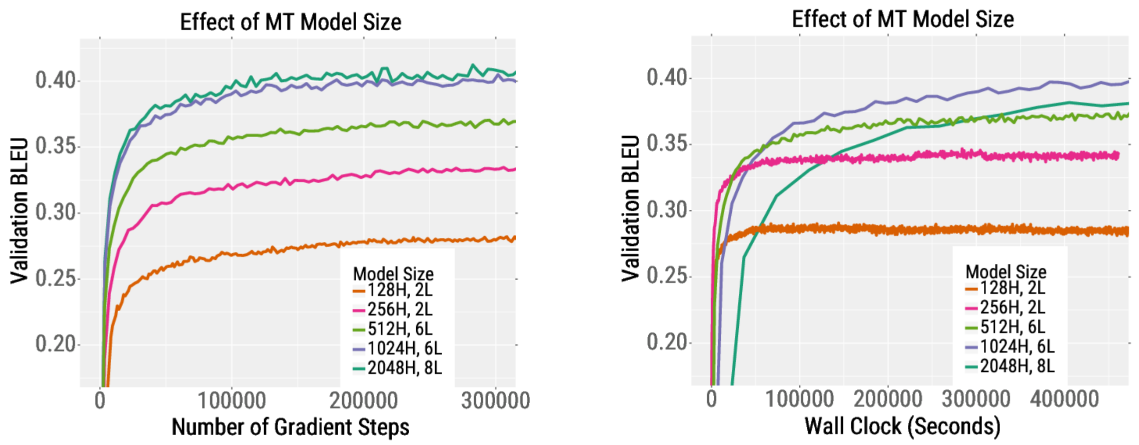


Figure 5. BLEU Scores on the English→French validation set (newstest2013) using models of different sizes. Larger models typically converge faster as a function of both iterations (left plot) and wall-clock time (right plot). When models become too large (2048H, 6L), they converge faster per iteration but their overhead on our limited hardware negates their convergence improvements.

Finetuning Setup and Compression Evaluation We focus on compressing the finetuned RoBERTa models as a case study. We train models of different sizes for 1,000,000 seconds,[5] finetune them on MNLI/SST-2, and then apply quantization/pruning. For evaluation, even though pruning and quantization will improve inference latency/throughput, quantifying these improvements is challenging because they are highly hardware-dependent. Instead, we follow past work and report the memory needed to store the model parameters (Thakker et al., 2019; Shen et al., 2020).

4.2. Larger Models Are More Robust to Quantization

We quantize every parameter, including the embedding matrix, but keep the model activations at full precision. We use floating point precisions in $\{4, 6, 8, 32\}$ bits (using lower than 4-bits resulted in severe accuracy loss). We apply quantization post-hoc which adds *no* additional time.

We quantize uniformly: the range of floats is equally split and represented by unsigned integers in $\{0, \ldots, 2^k - 1\}$, where k is the precision. We accomplish this by quantizing the weights W as:

$$W' = \mathsf{Clamp}(W, q_0, q_{2^k-1}),$$
$$W^I = \lfloor \frac{W' - q_0}{\Delta} \rceil, \text{ where } \Delta = \frac{q_{2^k-1} - q_0}{2^k - 1},$$
$$\mathsf{Quantize}(W) = \Delta W^I + q_0,$$

where $\mathsf{Clamp}()$ clamps all elements to the min/max range, W^I is a set of integer indices, $\lfloor \cdot \rceil$ is the round operator, Δ is the distance between two adjacent quantized points, and $[q_0, q_{2^k-1}]$ indicates the quantization range.

[5]We expect similar conclusions to hold for other budgets.

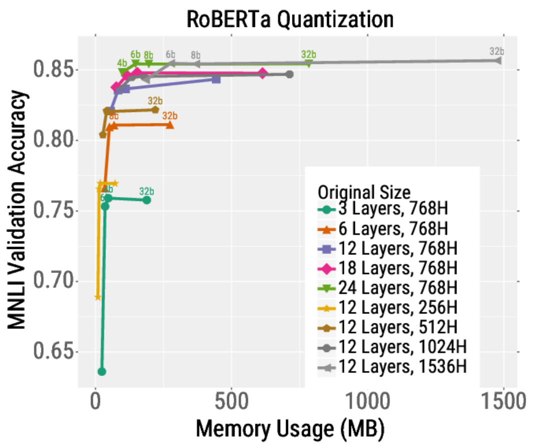
Results The quantization results for MNLI are shown on the left of Figure 6 (SST-2 results are in Appendix D). We plot each model's accuracy at different quantization levels as a function of its total memory usage. The larger models are more robust to quantization than the smaller models (the accuracy drop is smaller when the precision is reduced). Hence, the models which are trained using large parameter counts and then heavily quantized achieve the highest accuracy for almost all memory budgets.

4.3. Larger Models Are More Robust to Pruning

We use iterative magnitude pruning (Ström, 1997; Han et al., 2016): we iteratively zero out the smallest magnitude parameters and continue finetuning the model on the downstream task to recover lost accuracy.

Concretely, we consider models with sparsity levels of 15%, 30%, 45%, 60%, 75%, and 90%. We first find the 15% of weights with the smallest magnitude and set them to zero.[6] We then finetune the model on the downstream task until it reaches within 99.5% of its original validation accuracy or until we reach one training epoch. We then repeat this process—we prune another 15% of the smallest magnitude weights and finetune—stopping when we reach the desired sparsity level. The additional training overhead from this iterative process is small because the model typically recovers its accuracy in significantly less than one epoch (sometimes it does not require any retraining to maintain 99.5%). For example, pruning to 45% can be done with one or two additional epochs of finetuning on MNLI.

[6]It also may be possible to remove entire attention heads in addition to zeroing out weights (Michel et al., 2019; Voita et al., 2019). This may further improve our compression results.



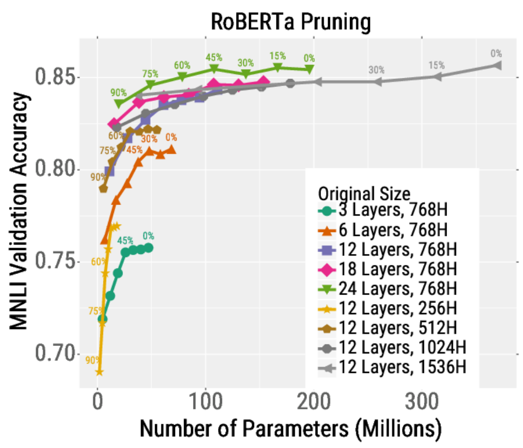


Figure 6. We first pretrain RoBERTa models of different sizes for the *same* total wall-clock time (larger models are trained for fewer steps). We then finetune each model on MNLI and compress them using quantization (left) and pruning (right). For most budgets (x-axis), the highest accuracy models are the ones which are trained large and then heavily compressed. The labels above each point indicate the compression amount (e.g., 4-bit quantization or 45% sparsity); we omit cluttered labels. SST-2 results are shown in Appendix D.

Results The pruning results for MNLI are shown in the right of Figure 6. We report the model's accuracy as a function of the total number of nonzero parameters.[7] The larger models can be pruned more than the smaller models without significantly hurting accuracy. Consequently, the large, heavily pruned models provide the best accuracy-efficiency trade-off. We find that deep networks are more robust to pruning than wider networks, e.g., the 24 Layer, 768H model outperforms the 12 Layer, 1536H model at most test budgets.

Combining Quantization and Pruning Results Pruning and quantization are complementary techniques for compressing Transformer models. We first prune models to various sparsity levels (e.g., 15%, 30%, etc.) and then apply varying amounts of quantization (e.g., 8-bit, 4-bit, etc.) to each model. In Figure 7 we plot combinations of pruning and quantization that lie at or near the Pareto frontier. Large models that are heavily compressed still provide the best trade-off between accuracy and efficiency when leveraging both pruning and quantization. A particularly strong compression method is to prune 30-40% of the weights and then quantize the model to 6-8 bits.

4.4. Convergence Does Not Affect Compressibility

Although larger Transformer models are more compressible, there is a confounding factor that our larger models are also less *converged* on the pretraining task. Is it the larger model size or the lack of convergence that causes the enhanced compressibility? We investigate this by finetuning RoBERTa models starting from different pretraining checkpoints (e.g., 3 epochs, 6 epochs, etc.) on MNLI. We then quantize the models to 4-bits.

[7]Since the reduction in memory from storing sparse matrices is highly dependent on the data structure used, we follow past work and report the number of nonzero model parameters (Luo et al., 2017; Li et al., 2017).

Figure 8 shows the results. Quantization is hardly affected by pretraining convergence—the drop in accuracy between the full precision and the 4-bit precision MNLI models is comparable as the pretrained model becomes more converged. Instead, the factor that determines compressibility is model size—the drop in accuracy is very large when compressing smaller models and vice versa.

5. When and Why Are Larger Models Better?

This section presents results and discussion on why larger Transformer models train faster and compress better.

5.1. Better Sample Efficiency With Larger Models

For larger models to train faster, they must converge faster (w.r.t. test error) per iteration. While there is a robust literature studying why larger models achieve better *final test accuracy*,[8] there is considerably less work exploring if and why larger models converge faster. One initial step in this direction is Arora et al. (2018a), who show that for deep *linear* neural networks, increasing depth can promote movement along directions already taken by the optimizer.

[8]Chiefly, this work seeks to reconcile the conflict between modern deep learning practice and the classical bias-variance trade-off. For instance, it studies forms of implicit regularization (Zhang et al., 2017; Belkin et al., 2018), characterizes the expressivity of deep models (Raghu et al., 2017; Lu et al., 2017), and bounds the neural network generalization error (Du et al., 2019; Arora et al., 2018b).

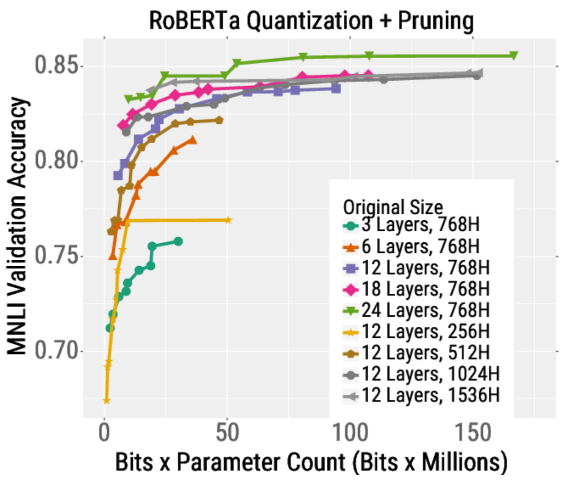


Figure 7. We combine pruning and quantization and find their gains to be complementary. The models which are trained large and then compressed are the best performing for each test-time budget.

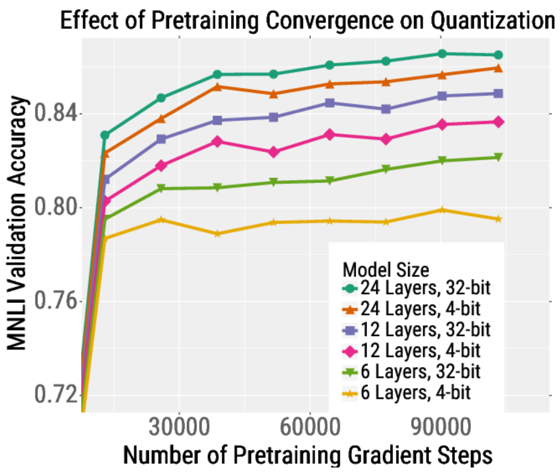


Figure 8. We disentangle whether *model size* or *pretraining convergence* causes the enhanced compressibility of larger models. We finetune RoBERTa models starting from different pretraining checkpoints on MNLI. We then quantize the models to 4-bits. Quantization is hardly affected by convergence—the drop in MNLI accuracy due to quantization is comparable as the pretrained model becomes more converged. Instead, the factor that determines compressibility is model size—the drop in accuracy is very large when compressing smaller models and vice versa.

Fast Minimization and the Role of Overfitting One empirical reason for the acceleration in convergence is that larger Transformer models minimize the training error faster. And, since the generalization gap is small for our tasks due to very large training sets, the larger models also converge faster w.r.t test error. In fact, the challenge in the MLM task is not *overfitting*, but instead, it is *fitting* the data—even 8 billion parameter models do not overfit to large pretraining corpora (Shoeybi et al., 2019).

When overfitting *is* a concern, larger models start to converge slower (w.r.t test error). We demonstrate this by randomly subsampling our pretraining dataset to 5% and 1% of its original size and training RoBERTa models of various sizes. When subsampling the data to 5% (top row of Figure 14 in Appendix A), the largest models do not improve on the training time of the smaller models (e.g., 12 layer RoBERTa trains just as fast as a 24 layer RoBERTa). Moreover, when the data is subsampled to 1% (bottom row of Figure 14), the largest models are worse in terms of perplexity due to overfitting. Thus, although our main conclusion that increasing model size accelerates convergence still holds for the smaller models (e.g., the 12 layer model outperforms the 3 layer one), overfitting causes it to break down for the largest models.

5.2. Manageable Compute Costs for Large Models

For larger models to train faster with respect to wall-clock time, their convergence improvements must not be negated by their slowdown in per-iteration time. Fortunately, parallel hardware (e.g., GPUs, TPUs) is usually not compute bound when training deep learning models. Instead, memory storage/movement is the limiting factor in image classification (Gomez et al., 2017), semantic segmentation (Chen et al., 2017), language modeling (Kitaev et al., 2020), and other tasks (Jain et al., 2020). Thus, larger models will more fully utilize the available compute, causing their slowdown to be sublinear. Moreover, when larger models cause hardware to run out of memory, gradient accumulation can trade-off memory for compute while still preserving the gains of large models, as shown in our experiments.

5.3. Smaller Compression Error for Larger Models

Large transformer models are more compressible than small transformer models.[9] Here, we present initial experiments to better understand why this occurs.

Quantization Error is Smaller for Larger Models We first measure the *quantization error*—the difference between the full-precision and low-precision weights—for the 4-bit RoBERTa models. On the left of Figure 9, we plot this value for models of varying depths (6, 12, and 24 layers) averaged across different Transformer modules (e.g., in-projection matrix of the self-attention). The mean and variance of the quantization error are smaller for deeper models.

Pruning Error is Smaller for Larger Models Similarly, we measure the *pruning error*—the difference between the

[9]Similar findings hold for large but sparse audio synthesis models (Kalchbrenner et al., 2018) and convolutional models for computer vision (Zhu & Gupta, 2018; Elsen et al., 2019; Evci et al., 2020; Kusupati et al., 2020).

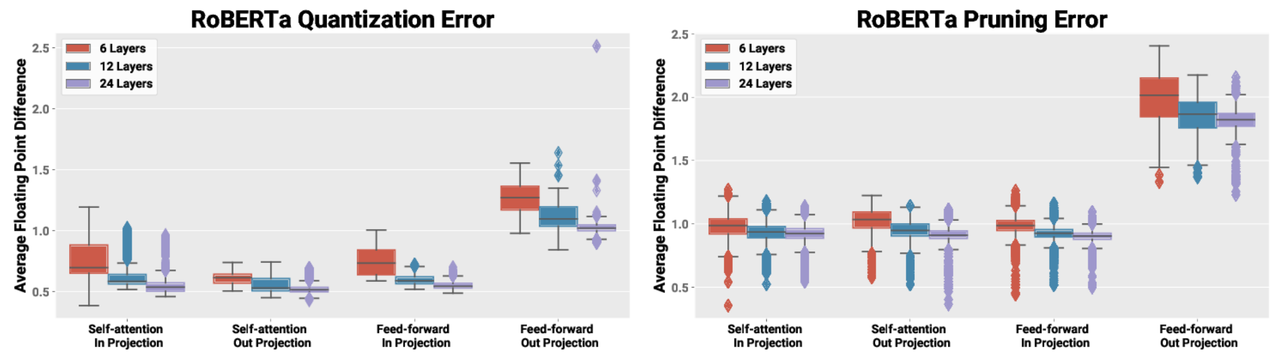


Figure 9. We finetune RoBERTa models of different sizes (6 layers, 12 layers, and 24 layers) on MNLI. We then quantize models to 4-bits or prune models to 60% sparsity. We plot the difference between the weights of the original and the quantized/pruned models averaged across different modules in the Transformer. The mean and variance of the weight difference after quantization (left) is consistently lower for the deeper models compared to the shallower models. The same holds for the difference after pruning (right). This shows that the larger model's weights are naturally easier to approximate with low-precision / sparse matrices than smaller models.

original weights and the sparse weights—for the 60% sparse RoBERTa models. The mean and variance of the pruning error are smaller for deeper models (Figure 9, right).

These two results show that the larger model's weights are more easily approximated by low-precision or sparse matrices. Interestingly, this phenomenon naturally occurs without directly optimizing for it; an area for future work is to study why these weight patterns emerge in larger models.

Connection to the Lottery Ticket Hypothesis Our compression findings have deep connections to recent conjectures such as the lottery ticket hypothesis (Frankle & Carbin, 2019). The lottery ticket hypothesis argues that larger models are preferable as they have a higher chance of finding a lucky initialization in one of their subnetworks. Our work shows that, for certain accuracies, as models become *increasingly large*, they contain *increasingly small* subnetworks which achieve that accuracy.

6. Related Work

Improving Training Speed and Efficiency There is a large body of work on accelerating model training, traditionally accomplished via improved optimizers (Nesterov, 1983; Kingma & Ba, 2015). More recent work improves training efficiency by modifying loss functions (Clark et al., 2020), model structures/sparsities (Louizos et al., 2018; Gong et al., 2019; Tan & Le, 2019), backpropagation storage requirements (Gruslys et al., 2016), or learning rate schedules (Loshchilov & Hutter, 2017; Li et al., 2020). We study the impact of model size, which is largely orthogonal to these other training efficiency improvements.

Scaling Model Training Another line of work scales model training to large amounts of distributed hardware and addresses the associated systems and machine learning challenges (Goyal et al., 2017; Ott et al., 2018; You et al., 2020). Our work instead looks to choose the optimal model size for a fixed (small) hardware budget. Future work can study whether our conclusion that large models are more compute-efficient also holds in this highly-distributed setting, where the "budget" is extremely large.

Hyperparameter Tuning and AutoML In our work, we have an initial setting for the hyperparameters and optimize the model size. However, good initial models and hyperparameters are unknown when approaching *new* problems. For these cases, the optimal training strategy must consider the cost of experimenting with different architectures and hyperparameters; future work can study the effect of model size in this setting. More generally, our findings may impact the design of automated methods for solving/optimizing machine learning problems (Feurer et al., 2015; Zoph & Le, 2017; Jaderberg et al., 2017). In particular, the compute-efficiency of these methods may improve by following our *train large, then compress* methodology.

Training Efficiency of Large Models Recent and concurrent work also considers the impact of model size on the compute efficiency of training. Raffel et al. (2019) show that training a 4x larger Transformer model is a good usage of 4x more compute. Ardalani et al. (2019) show that larger RNN models take fewer gradient iterations to converge but do not consider that larger models are faster when adjusting for wall-clock time. In concurrent work, Kaplan et al. (2020) study the impact of model size on the training efficiency of Transformer language models. They make similar conclusions that large, undertrained models are superior to small, well-trained models. Our work differs in that we study machine translation and the impact of training large models on downstream tasks (model finetuning and compression).

7. Conclusion and Future Work

We studied the impact of Transformer model size on the efficiency of training and inference. We show that increasing model width and depth accelerates convergence in terms of both gradient steps and wall-clock time. Moreover, even though large models appear less efficient during inference, we demonstrate that they are more robust to compression. Therefore, we conclude that the best strategy for resource-constrained training is to *train large models and then heavily compress them*.

In the future, we will examine these conclusions on more domains such as computer vision. Moreover, we look to answer the questions that are raised by our results: *why* do larger transformer models train fast and compress well, how does model size impact overfitting and hyperparameter tuning, and more generally, what other common design decisions should be rethought in the compute-efficient setting?

Acknowledgements

This research was supported by the Berkeley RISE Lab. We would like to thank the Google Cloud TPU team for their hardware support. We are also grateful to Shi Feng, Yang Liu, Suchin Gururangan, Nelson Liu, the members of Berkeley NLP, and the members of the Berkeley RISE Lab for their valuable feedback.

References

Ardalani, N., Hestness, J., and Diamos, G. Empirically characterizing overparameterization impact on convergence. *OpenReview: S1lPShAqFm*, 2019.

Arora, S., Cohen, N., and Hazan, E. On the optimization of deep networks: Implicit acceleration by overparameterization. In *ICML*, 2018a.

Arora, S., Ge, R., Neyshabur, B., and Zhang, Y. Stronger generalization bounds for deep nets via a compression approach. In *ICML*, 2018b.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. In *NeurIPS*, 2016.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. Reconciling modern machine learning and the bias-variance trade-off. In *PNAS*, 2018.

Brock, A., Donahue, J., and Simonyan, K. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. In *TPAMI*, 2017.

Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*, 2020.

Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., and Stoica, I. Clipper: A low-latency online prediction serving system. In *NSDI*, 2017.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. Universal transformers. In *ICLR*, 2019.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

Dong, Z., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. HAWQ: Hessian aware quantization of neural networks with mixed-precision. In *ICCV*, 2019.

Du, S. S., Zhai, X., Poczos, B., and Singh, A. Gradient descent provably optimizes over-parameterized neural networks. In *ICLR*, 2019.

Elsen, E., Dukhan, M., Gale, T., and Simonyan, K. Fast sparse convnets. *arXiv preprint arXiv:1911.09723*, 2019.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *ICML*, 2020.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. Efficient and robust automated machine learning. In *NeurIPS*, 2015.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

Gomez, A. N., Ren, M., Urtasun, R., and Grosse, R. B. The reversible residual network: Backpropagation without storing activations. In *NeurIPS*, 2017.

Gong, L., He, D., Li, Z., Qin, T., Wang, L., and Liu, T. Efficient training of BERT by progressively stacking. In *ICML*, 2019.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Gruslys, A., Munos, R., Danihelka, I., Lanctot, M., and Graves, A. Memory-efficient backpropagation through time. In *NeurIPS*, 2016.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

Hénaff, O. J., Srinivas, A., Fauw, J. D., Razavi, A., Doersch, C., Eslami, S. M. A., and van den Oord, A. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272*, 2019.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Jain, P., Jain, A., Nrusimha, A., Gholami, A., Abbeel, P., Keutzer, K., Stoica, I., and Gonzalez, J. E. Checkmate: Breaking the memory wall with optimal tensor rematerialization. In *MLSys*, 2020.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, 2017.

Kalchbrenner, N., Elsen, E., Simonyan, K., Noury, S., Casagrande, N., Lockhart, E., Stimberg, F., Oord, A. v. d., Dieleman, S., and Kavukcuoglu, K. Efficient neural audio synthesis. In *ICML*, 2018.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. In *ICLR*, 2020.

Kusupati, A., Ramanujan, V., Somani, R., Wortsman, M., Jain, P., Kakade, S., and Farhadi, A. Soft threshold weight reparameterization for learnable sparsity. In *ICML*, 2020.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. ALBERT: A lite BERT for self-supervised learning of language representations. In *ICLR*, 2020.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *NeurIPS*, 1990.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. In *ICLR*, 2017.

Li, M., Yumer, E., and Ramanan, D. Budgeted training: Rethinking deep neural network training under resource constraints. In *ICLR*, 2020.

Liu, L., Wang, H., Lin, J., Socher, R., and Xiong, C. Attentive student meets multi-task teacher: Improved knowledge distillation for pretrained models. *arXiv preprint arXiv:1911.03588*, 2019a.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019b.

Loshchilov, I. and Hutter, F. SGDR: Stochastic gradient descent with warm restarts. In *ICLR*, 2017.

Louizos, C., Welling, M., and Kingma, D. P. Learning sparse neural networks through L_0 regularization. In *ICLR*, 2018.

Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. The expressive power of neural networks: A view from the width. In *NeurIPS*, 2017.

Luo, J.-H., Wu, J., and Lin, W. ThiNet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

McCandlish, S., Kaplan, J., Amodei, D., and Team, O. D. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

Metz, C. Building an AI chip saved Google from building a dozen new data centers. *Wired*, 2017.

Michel, P., Levy, O., and Neubig, G. Are sixteen heads really better than one? In *NeurIPS*, 2019.

Nesterov, Y. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. In *Soviet Mathematics Doklady*, 1983.

Ott, M., Edunov, S., Grangier, D., and Auli, M. Scaling neural machine translation. In *WMT*, 2018.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. Fairseq: A fast, extensible toolkit for sequence modeling. In *NAACL Demo*, 2019.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. BLEU: a method for automatic evaluation of machine translation. In *ACL*, 2002.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., and Dickstein, J. S. On the expressive power of deep neural networks. In *ICML*, 2017.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. In *NeurIPS EMC^2 Workshop*, 2019.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. Green AI. *arXiv preprint arXiv:1907.10597*, 2019.

See, A., Luong, M.-T., and Manning, C. D. Compression of neural machine translation models via pruning. In *CoNLL*, 2016.

Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., et al. Mesh-TensorFlow: Deep learning for supercomputers. In *NeurIPS*, 2018.

Shen, S., Dong, Z., Ye, J., Ma, L., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. Q-BERT: Hessian based ultra low precision quantization of BERT. In *AAAI*, 2020.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-LM: Training multi-billion parameter language models using GPU model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

Ström, N. Sparse connection and pruning in large dynamic artificial neural networks. In *EUROSPEECH*, 1997.

Sun, S., Cheng, Y., Gan, Z., and Liu, J. Patient knowledge distillation for BERT model compression. In *EMNLP*, 2019.

Tan, M. and Le, Q. V. EfficientNet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

Thakker, U., Beu, J., Gope, D., Zhou, C., Fedorov, I., Dasika, G., and Mattina, M. Compressing RNNs for IOT devices by 15-38x using kronecker products. *arXiv preprint arXiv:1906.02876*, 2019.

Turc, I., Chang, M.-W., Lee, K., and Toutanova, K. Well-read students learn better: The impact of student initialization on knowledge distillation. *arXiv preprint arXiv:1908.08962*, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *NeurIPS*, 2017.

Vaswani, A., Bengio, S., Brevdo, E., Chollet, F., Gomez, A. N., Gouws, S., Jones, L., Kaiser, Ł., Kalchbrenner, N., Parmar, N., et al. Tensor2Tensor for neural machine translation. In *AMTA*, 2018.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *ACL*, 2019.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*, 2019a.

Wang, Q., Li, B., Xiao, T., Zhu, J., Li, C., Wong, D. F., and Chao, L. S. Learning deep transformer models for machine translation. In *ACL*, 2019b.

Williams, A., Nangia, N., and Bowman, S. R. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL*, 2018.

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. Large batch optimization for deep learning: Training BERT in 76 minutes. In *ICLR*, 2020.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.

Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. In *ICLR Workshop Track*, 2018.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *CVPR*, 2015.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

A. Additional Training Curves

A.1. Training Cost Using FLOPs

In Figure 10, we plot selected learning curves from the main text as a function of FLOPs rather than seconds. We compute FLOPs using the code provided by Clark et al. (2020).

A.2. The Impact of Batch Size

Figure 13 shows the learning curves associated with different batch sizes. Table 1 shows the learning rates associated with each batch size. We use the hyperparameters from Liu et al. (2019b) as a starting point and then lightly tune them.

Batch Size	Learning Rate
256	.0002
2048	.001
4096	.00125
8192	.0015
16384	.001875

Table 1. The learning rate for each batch size in Figure 13.

A.3. The Impact of Dataset Size

Figure 14 shows the learning curves for models trained using 5% and 1% of the training data.

B. Finetuning Models of Different Sizes

Table 2 shows that models with more parameters are not harder to finetune.

Model	Perplexity	MNLI	SST-2
12-layer, 768H	4.3	84.3	93.0
18-layer, 768H	4.1	85.4	92.6
24-layer, 768H	4.0	85.2	93.1
12-layer, 768H	4.3	84.3	93.0
12-layer, 1024H	3.9	85.5	93.2
12-layer, 1536H	4.3	85.1	93.8

Table 2. We train RoBERTa models of different sizes and stop them at roughly the same pretraining perplexity (the bigger models are trained for less wall-clock time). We then finetune each model on MNLI and SST-2. All models reach comparable accuracies (in fact, the big models often outperform small ones), which shows that larger models are not harder to finetune.

C. Negative Results: Layer Sharing

Sharing weights across transformer layers can provide a small or negligible degradation in final performance (Lan et al., 2020; Dehghani et al., 2019) while providing a reduction in memory consumption. In addition, models with shared layers are slightly faster to execute because they require less memory movement and reduced inter-device communication. Similar to Lan et al. (2020), we experiment with two types of layer sharing: sharing all layers and sharing only the attention layers.

Sharing layers reduces the maximum memory requirements, especially for small batch sizes. For example, sharing all the layers of a RoBERTa model with batch size 32 reduces total memory usage by 41%. However, both forms of sharing lead to slower training convergence and thus worse performance in the resource-constrained setting (Figure 11). Consequently, we do not recommend sharing layers for compute-efficient training or inference of transformers.

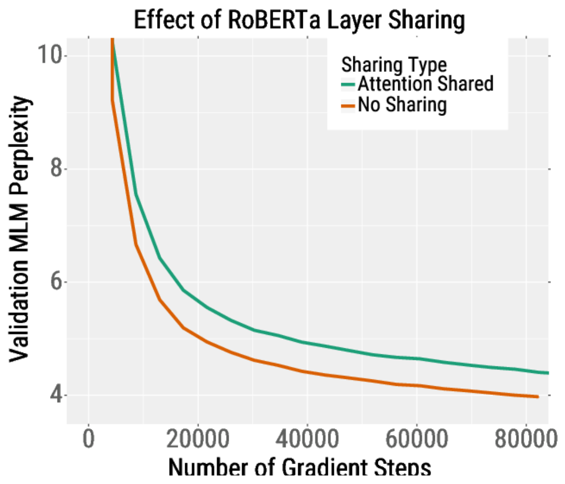


Figure 11. Sharing attention layers reduces the maximum memory consumption of RoBERTa but causes slower convergence and worse final accuracy.

D. Compression Results for SST-2

We follow Liu et al. (2019b) and report results on SST-2 (Socher et al., 2013) in addition to MNLI. Since the SST-2 dataset is smaller than MNLI it requires a more significant tuning of the finetuning hyperparameters. We tune the batch size in $\{16, 32, 64\}$, the learning rate in $\{5e{-}4, 3e{-}4, 1e{-}4\}$, the seed which controls the classifier initialization and training data shuffling in $\{100, 300, 500\}$, and the dropout in $\{0.1, 0.2, 0.3\}$. We choose the best value using the validation set for each model size. We then perform quantization, pruning, and quantization and pruning on all finetuned models. Similar to MNLI, the bigger models provide the highest accuracy for a given test budget (Figure 12).

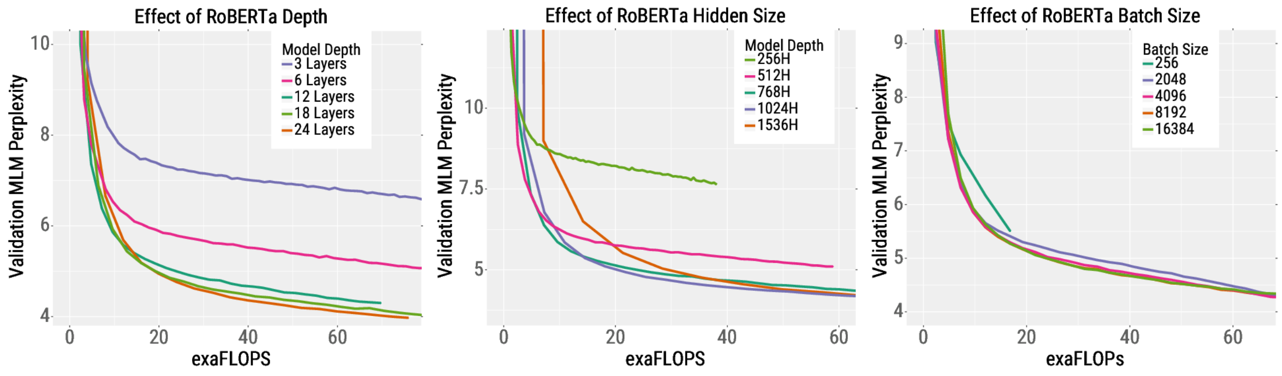


Figure 10. Floating Point Operations. We show Figures 2, 4, and 13 in terms of exaFLOPs instead of wall-clock time. Bigger models achieve better results than smaller models using the same number of floating point operations.

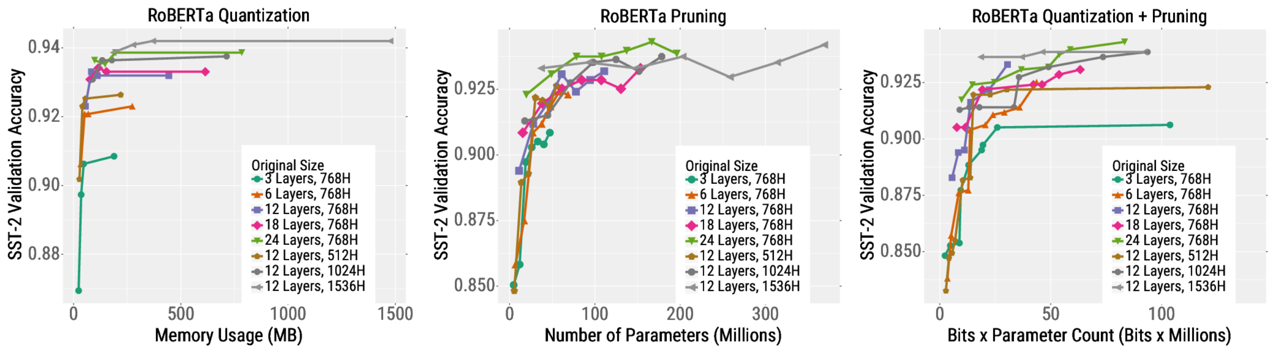


Figure 12. Compression for SST-2. For most budgets (x-axis), the highest accuracy SST-2 models are the ones which are trained large and then heavily compressed. We show results for quantization (left), pruning (center), and quantization and pruning (right).

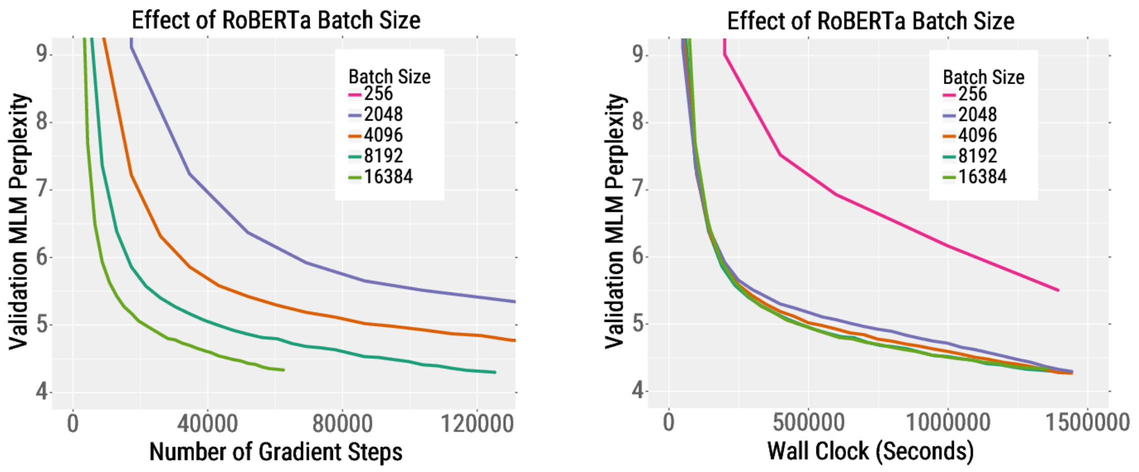


Figure 13. Increasing the batch size and the associated learning rate accelerates convergence in terms of gradient steps. However, increasing the batch size beyond 2048 provides only marginal improvements with respect to wall-clock time. Note that the wall-clock time includes the cost of accumulating gradients on a single machine (see Section 2.2). In other words, beyond a certain point increasing the batch size only provides speedups when additional hardware is available. The 256 batch size result is far to the right in the left plot.

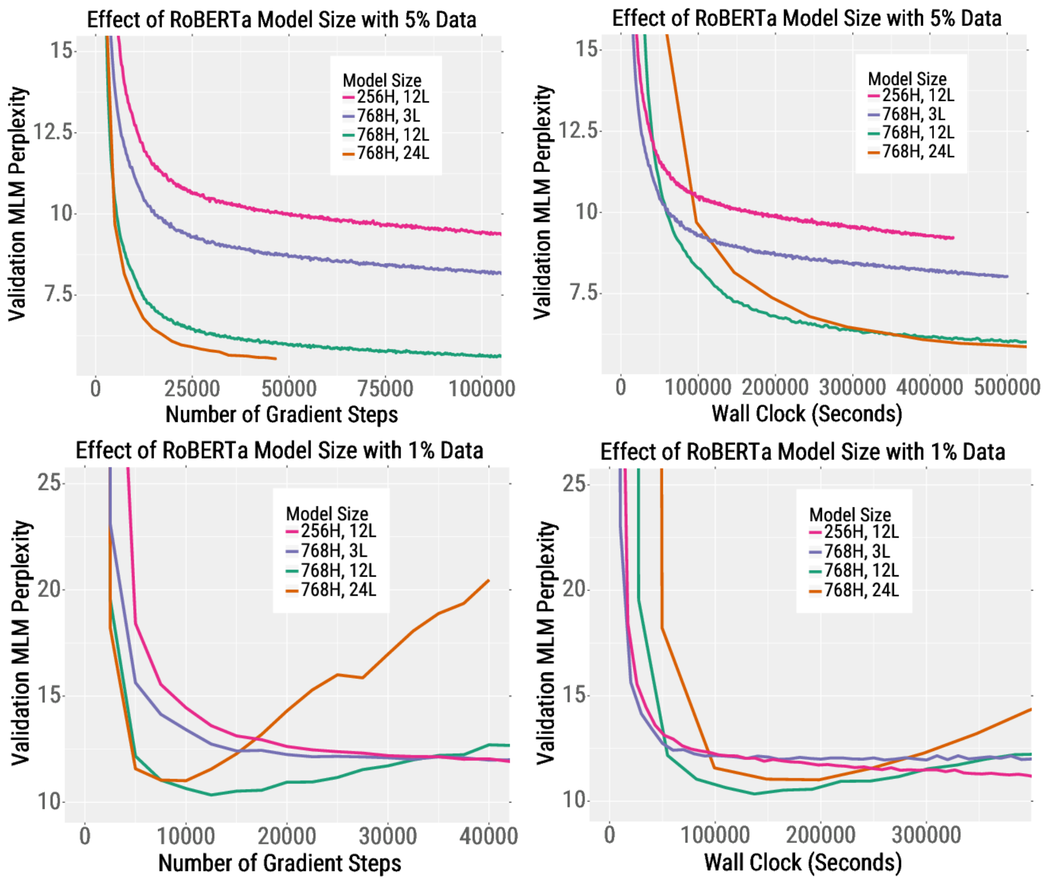


Figure 14. Effect of Smaller Datasets. In our experiments on the full dataset (see main text), the largest models we trained are always faster in terms of wall-clock time. However, when subsampling the data to 5% (top row), the biggest models do not improve on the speed of the smaller models (e.g., compare 24 Layer RoBERTa and 12 Layer RoBERTa). When the data is subsampled to 1% (bottom row), the bigger models are *worse* in terms of perplexity due to overfitting. This illustrates that the optimal model size depends on the dataset size.