Block Pruning For Faster Transformers

François Lagunas, Ella Charlaix, Victor Sanh, Alexander M. Rush
Hugging Face
{francois, ella, victor, sasha}@huggingface.co

Abstract

Pre-training has improved model accuracy for both classification and generation tasks at the cost of introducing much larger and slower models. Pruning methods have proven to be an effective way of reducing model size, whereas distillation methods are proven for speeding up inference. We introduce a block pruning approach targeting both small and fast models. Our approach extends structured methods by considering blocks of any size and integrates this structure into the movement pruning paradigm for fine-tuning. We find that this approach learns to prune out full components of the underlying model, such as attention heads. Experiments consider classification and generation tasks, yielding among other results a pruned model that is a 2.4x faster, 74% smaller BERT on SQuAD v1, with a 1% drop on F1, competitive both with distilled models in speed and pruned models in size.

1 Introduction

Pre-trained transformer models are the standard for NLP tasks in both classification and generation tasks (Devlin et al., 2019; Lewis et al., 2020). The recent trend is for models to continue to grow in size while yielding improved performance on standard benchmarks (Rosset, 2020). This development highlights the need to reduce the storage size and increase the efficiency of pre-trained models.

Pruning methods have shown to be extremely effective at reducing the storage size of models fine-tuned for a specific task. Approaches such as magnitude pruning (Han et al., 2015), L0 regularization (Louizos et al., 2018), lottery ticket hypothesis (Frankle and Carbin, 2018), diff pruning (Guo et al., 2020), and movement pruning (Sanh et al., 2020) have demonstrated remarkable reductions in model size. Movement pruning produces 77% savings in parameter storage for a 1% drop in accuracy on SQuAD v1.1. However, these models yield very little actual efficiency benefits, as to run them in standard hardware often requires reconstructing the original dense shape.

On the other hand distillation methods have been more effective at producing faster models as has been shown by DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2019) or MobileBERT (Sun et al., 2020). These approaches utilize targeted distillation to produce smaller models with a dense structure that is fast on standard hardware. However without careful engineering and size selection these models are much larger than pruned ones.

In this work, we target closing this gap through block pruning. Unlike pruning individual parameters, this approach encourages pruning that can be optimized on dense hardware. It is a less rigid approach than row or column-based pruning typically used in structured approaches (McCarley, 2019), which have been difficult to apply effectively to transformers. We integrate this approach with Movement pruning (Sanh et al., 2020), a simple method for pruning pre-trained models during fine-tuning. The final method[1] has few additional hyperparameters or training requirements.

Experiments consider a large variety of different benchmark datasets comparing accuracy and efficiency. We find a surprising result that despite utilizing sub-row square blocks during training, the approach learns to eliminate full components of the model, effectively dropping a large number of attention heads. This effect allows the model to achieve speedups even beyond standard structured pruning of feed-forward layers. Results show a 2.4x speedup on SQuAD v1.1 with a 1% drop of F1, and a 2.3x speedup on QQP with a 1% loss of F1. Experiments on summarization also show a 1.39x speedup for an average of 2 points drop on all ROUGE metrics on CNN/DailyMail, and for a reduction of decoder weights of 3.5x.

[1] Available at https://github.com/huggingface/nn_pruning

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 10619–10629
November 7–11, 2021.

2 Related Work

There has been a growing interest in the compression of pre-trained language models. We consider three varieties of methods: distillation, pruning, and structured pruning.

Knowledge distillation, introduced by Hinton et al. (2015), is a popular compression technique. Researchers have applied this method to a variety of NLP models (Tang et al., 2019; Sun et al., 2019; Turc et al., 2019). Distillation has been used to obtain significantly smaller BERT models achieving competitive performances. Sanh et al. (2019) distills BERT into shallower students during the pre-training stage and optionally during the fine-tuning stage. MobileBERT (Sun et al., 2020) and TinyBERT (Jiao et al., 2019) are obtained thanks to a layer-wise distillation strategy. While the distillation of former is task-agnostic, the one used to obtain the latter is task-specific.

Other previous work has focused on unstructured pruning (LeCun et al., 1989; Han et al., 2015; Frankle and Carbin, 2018). When targeting transformer models, it is typical to select the weights to prune based on their magnitude (Gordon et al., 2020), or by computing an importance score using a first-order method (Sanh et al., 2020). While these methods allow for a significant reduction in model size, specialized hardware is required to make use of the resulting unstructured sparse matrices in order to speed up inference.

In contrast, structured pruning removes coherent groups of weights (Murray and Chiang, 2015; See et al., 2016; Joulin et al., 2016; Fan et al., 2020; Sajjad et al., 2020). Recent works (Michel et al., 2019; Voita et al., 2019) show that some heads can be removed without significant degradation in performance, leading to the conclusion that most heads provide redundant information. Other authors have worked on combining matrix factorization and weight pruning. While Mao et al. (2020) combine SVD-based matrix factorization with unstructured pruning, Wang et al. (2019) use structured pruning in order to reduce the rank. Related to our approach, Kim and Awadalla (2020) and McCarley (2019) both apply structured pruning on the heads of the multi-head attention (MHA) and on the inner-layer nodes of the feed-forward network (FFN). The former uses predefined pruning ratios, shared across all layers, in order to select the modules to prune after sorting them given an importance score. McCarley (2019) compares different methods to compute the prunable module masks and find L0 regularization to perform the best.

3 Background

Starting with a transformer model with parameters θ, our goal is to produce a set of parameters θ' that are both fine-tuned for a specific end-task and smaller in such a way that inference can be efficiently computed on parallel hardware.

The two largest lines in the transformer parameter budget are the feed-forward network sub-layer (FFN) and the multi-head attention sub-layer (MHA). The FFN parameters consist of two matrices ($\mathbf{W_1}$ and $\mathbf{W_2}$) of transposed shape $\mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$ and $\mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$ where d_{model} is the hidden size and $d_{\text{ff}} \gg d_{\text{model}}$ is the inner size. These are used in the standard fashion by the network. The MHA parameters consist of 4 projection matrices ($\mathbf{W_q}$, $\mathbf{W_k}$, $\mathbf{W_v}$ and $\mathbf{W_o}$) of size $\mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ (query, key, value, out). These are used to project the hidden vector to and from the component attention parts. In implementations, this projection is made with the matrices in their folded tensor form $\mathbb{R}^{n_{\text{heads}} \times \frac{d_{\text{model}}}{n_{\text{heads}}} \times d_{\text{model}}}$ where n_{heads} is the number of attention heads.

In standard fine-tuning, starting from θ, we optimize the loss $\mathcal{L}$ (for instance, cross-entropy for classification):

$$\arg\min_{\theta'} \mathcal{L}(\theta')$$

Score-based pruning methods (Ramanujan et al., 2019) modify the model by introducing score parameters S for each parameter i and replace the original parameter matrices with a masked version $\mathbf{W}' = \mathbf{W} \odot M(\mathbf{S})$. For instance, in the simplest version of magnitude pruning, the mask would just zero-out parameters with low absolute values.

Movement pruning (Sanh et al., 2020) is a score-based pruning approach that encourages the model to optimize these score parameters. Specifically, we focus on the soft-movement variant of movement pruning that sets $M(\mathbf{S}) = \mathbf{1}(\mathbf{S} > \tau)$ for a threshold parameter τ, and optimizes a regularized objective,

$$\arg\min_{\theta', S} \mathcal{L}(\theta') + \lambda \|\sigma(\mathcal{S})\|$$

where λ is a hyper-parameter, $\|A\| = \sum_{i,j} A_{i,j}$ and σ is the sigmoid function.

This pruning objective encourages the model to fine-tune the parameters while lowering the scores of unimportant parameters and thus encouraging more sparsity. In order to train through the threshold, a straight-through estimator (Bengio et al., 2013) is used.

Movement pruning, combined with distillation, has shown to be a very effective method to reduce the number of parameters in an existing model yielding 94% pruning in our tests for a F1 of 87.5 on SQuAD v1.1 (BERT-base is 88.5). This results in significantly smaller models than distillation alone. However, even with this sparsity level, the model is not substantially faster when run on most standard hardware that cannot significantly take advantage of this style of sparse matrix-vector product.

4 Model: Block Movement Pruning

In this work, we extend movement pruning to work on blocks of local parameters. Specifically, each matrix in the transformer is partitioned into fixed-sized blocks. This setting goes beyond the arbitrary pruning of unstructured methods, with the goal of encouraging the data locality closer to what would be needed for efficiency.[2]

Our approach is extremely simple. For each parameter matrix $\mathbf{W} \in \mathbb{R}^{M \times N}$, we assume a fixed-sized block structure (M', N'). Each of these blocks acts as an individual group in the regularization with a shared score parameter derived from the corresponding score matrix $\mathbf{S} \in \mathbb{R}^{M/M' \times N/N'}$. Computing the masked weight is done by expanding the thresholded values, i.e.

$$W'_{i,j} = W_{i,j} * M(\mathbf{S})_{\lceil i/M' \rceil, \lceil j/N' \rceil}$$

As in past work, this model is trained with distillation to match the performance of a teacher model.

Unlike other distillation approaches that require fully specifying the new model structure, our method only requires the size and shapes of the blocks, i.e. the set of (M', N') for each parameter matrix in the model. If blocks are too large, then they are difficult to prune, but if they are too small they do not support efficient inference.

[2]Linear algebra libraries perform matrix multiplication using large blocks, typically 128*64. At a micro level those machines are typically 32 ways SIMD, and memory is loaded by large contiguous chunks to maximize bandwidth. Unstructured sparsity is hard to implement with dense algebra performance on GPUs. Data locality is important on CPU too, but in a more limited way.

To reduce the search space, we will limit ourselves to test $(M', N')^{\text{att}}$ and $(M', N')^{\text{ff}}$: the same block size will be used for all layers for attention weights $\mathbf{W_q}$, $\mathbf{W_k}$, $\mathbf{W_v}$ and $\mathbf{W_o}$ on one hand, and for the feed-forward weights $\mathbf{W_1}$ and $\mathbf{W_2}$ on the other hand. We split the movement pruning regularization term into:

$$\lambda_{\text{att}} \|\sigma(S_{\text{att}})\| + \lambda_{\text{ffn}} \|\sigma(S_{\text{ffn}})\|$$

This allows us to take into account the difference in terms of gradient received by the score parameters.

To reduce further the search space, we will test on two kinds of blocks:

- $(32, 32)$: square blocks (*Block*)
- $(1, d_{\text{model}})$ and $(d_{\text{model}}, 1)$: dimension pruning on paired FFN rows and columns (*Dim*)

These block sizes allow for efficient models: blocks of size at least $(16, 16)$ are efficient to compute with appropriate GPU kernels, whereas full rows, columns or heads can be entirely removed from the matrix: the remaining matrix is then dense.

We also include two additional baseline block types used to verify the approach:

- $(2^n, 2^n), n \in [2, 5]$: smaller power of two square block sizes to study the impact of size on performance (*Block*)
- $(\frac{d_{\text{model}}}{n_{\text{heads}}}, d_{\text{model}})$: for attention heads (*Heads*)

The first considers small blocks, and the second considers very large functional blocks.

5 Experimental Setup

We conduct experiments on five (English) tasks commonly used to evaluate pre-trained language models: question answering (SQuAD v1.1 Rajpurkar et al., 2016) and (SQuAD v2 Rajpurkar et al., 2018), natural language inference (MNLI Williams et al., 2018), sentence similarity (QQP Chen et al., 2018), sentiment classification (SST-2 Socher et al., 2013) and abstractive summarization (CNN/DailyMail Hermann et al., 2015). These datasets respectively contain 87k, 130k, 392k, 363k, 67k and 287k training examples, and are downloaded from the Hugging Face datasets hub. SQuAD is formulated as a span-extraction task, MNLI and QQP are sentence pairs classification tasks, SST-2 is a sentence classification task and

CNN/DailyMail ("CNN") is formulated as a conditional generation task. We report the performance on the development set as measured by the accuracy for MNLI and SST-2, F1 for QQP, the exact match (EM) and F1 for SQuAD and ROUGE for CNN/DailyMail.

We experiment with task-specific pruning of transformer language models. We use BERT (Devlin et al., 2019) (an encoder-only Transformer language model with 110M parameters, among which 85M are part of the linear layers present in the Transformer layers) for sentence classification and question answering (340M and 227M respectively for BERT-large), and BART (Lewis et al., 2020) (an encoder-decoder language model with 139M parameters, among which 99M are part of the linear layers present in the Transformer layers) for summarization (406M and 353M for BART-large).

We compare against several baselines. Movement pruning is a fully unstructured approach and gives an upper bound on the sparsity trade-offs we hope to achieve, even if it provides little speed benefit. We also compare our results against state-of-the-art approaches developed for fast inference of transformer-based language models. DistilBERT (Sanh et al., 2019) is obtained by distilling through pre-training a pre-trained BERT into a smaller model. TinyBERT (Jiao et al., 2019) distills a fine-tuned model while using data augmentation. MobileBERT (Sun et al., 2020) is the result of a large architecture search. dBART (Shleifer and Rush, 2020) is obtained by arbitrarily copying equally spaced layers of a large model to a smaller one. To measure inference speed on GPU, we use a 24GB 3090 RTX and an Intel i7 CPU, using a large batch size (128) for evaluation and using PyTorch CUDA timing primitives. We measure the speed of other models in this same setup. Results may be different from original papers, as latency and throughput characteristics are different for each platform. We also provide the number of parameters in the linear layers of the Transformer layers for each of our models and for the reference ones: as the linear layers represent most of the FLOPS, this is a good proxy for the computation required and to some extent for the compute time, when the model characteristics are equivalent.

Resources and Reproducibility

We are using a minimal set of hyperparameters. The ratio of $\lambda_{\mathbf{att}}$ and $\lambda_{\mathbf{ffn}}$ is fixed by the relative sizes. We performed a few experiments with different values fixed manually for these parameters, but their influence is minor.

Method	MHA	FFN	Teacher
Block	Block	Block	Base
Hybrid	Block	Dim	Base
Hybrid NT	Heads	Dim	None
Struct	Heads	Dim	Base
Hybrid Filled	Heads	Dim	Base
Hybrid Filled LT	Heads	Dim	Large

Table 1: Summary of pruning methods. Dim blocks correspond to row and column blocks for the FFN.

The main hyperparameter is the number of training epochs. For SQuAD v1.1, we are using 20 epochs instead of typically 2 for BERT models. This means a fine-tuning is taking about 12h with our method instead of 45mn with a standard fine-tuning setup. This number has to be large enough to let pruning happen slowly enough for a given task. A warming up phase and a post-pruning cool-down phase are helpful, but their exact length has not a large impact on final performance. We believe the training time is less important than the inference time for energy consideration, as inference is performed repeatedly. Our method is optimizing inference by a large factor: the training energy is potentially recouped by a large margin with inference savings.

Finally, the checkpoints created during the experiments are available on an AWS S3 bucket, with their metadata and training parameters, totaling 3TB of data, to facilitate reproduction of our results and to make it possible to study further the behavior of those models. Code for experiments, analysis, and tools to prepare the present paper are available on GitHub (see Appendix A).

Pruning Methods

The pruning approaches are shown in Table 1.

Block pruning use square block sizes throughout all the linear layers, as an extension of the original movement pruning for which the block size is 1.

Hybrid pruning jointly removes hidden dimensions in feed-forward layers $\mathbf{W_1}$ and $\mathbf{W_2}$, using movement pruning to create the dimension mask. This corresponds to full rows or columns in the parameter matrices. The pruned $\mathbf{W'_1}$ and $\mathbf{W'_2}$ can then be "compacted" to become fully dense: we

perform dense operations on cropped matrices. For the attention layers, pruning only some rows or columns in $\mathbf{W_q}$, $\mathbf{W_k}$, $\mathbf{W_v}$ and $\mathbf{W_o}$ can not be practically exploited. This is because the structure of the computation makes the additional cost of resizing the tensor inefficient. We, therefore, use square block pruning on the attention layer, with a block size of $(32, 32)$ which showed the best trade-off between performance and accuracy.

Struct pruning uses the same methods for FFN layers but aims to remove model attention heads directly. To do so, we choose a block size on attention that equals the head size while still using the same soft movement pruning strategy. For this approach, we use a λ_{att} equals to $1/32$, as there are 32 times more parameters than in an attention block than in a feed-forward dimension.

When Block Pruning does not fully remove a component such as an attention head, as shown in Figure 1, we cannot speed up the model. But we can reclaim some of the performance at no speed cost and at marginal cost on sparsity by making use of those zero weights.

Hybrid Filled pruning allows the model to reinitialize these reclaimed weights uniformly at random and continue fine-tuning the smaller model for a few steps. We also explore "rewinding" (Frankle and Carbin, 2018) by identifying weights that should not be pruned (because they are part of a non-empty attention head) and re-fine-pruning the pre-trained model: the first run marks the attention heads that were not pruned, and the second uses this information to create a positive mask of weights that are protected from pruning. We did not find a significant difference between the two methods. The results presented here do not use rewinding.

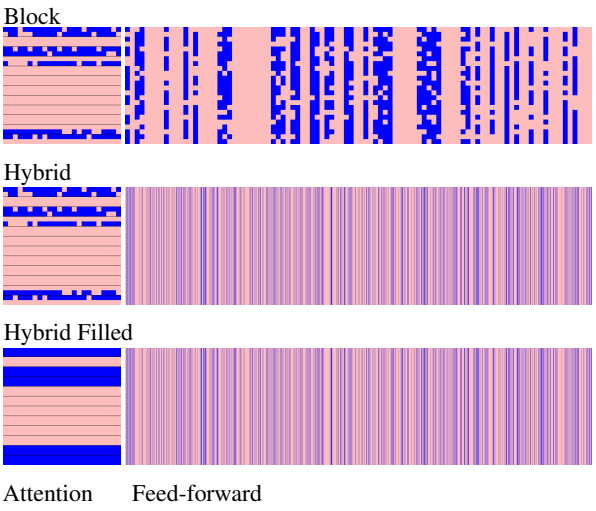

Figure 1: Pruning patterns on SQuAD v1.1: blue is preserved, pink is pruned. Attention heads are delimited for clarity.

6 Experiments

Main Results We begin by observing the high-level impact of the different pruning methods. Figure 1 shows the effect on attention and feed-forward layers for the different block pruning methods. We find that all the different block sizes learn to prune out entire dimensions in the FFN layers. Interestingly we find that the block methods can also learn to remove entire heads from the MHA. This pruning pattern makes it possible to remove entire heads from the model during inference. For this reason, we focus on the Hybrid approach as our main method, which can both eliminate feed-forward dimensions while using blocks to remove attention heads gradually.

Results on SQuAD are shown in Figure 2, which compares our approach for speed and density to baseline BERT-Base tuned models such as TinyBERT-6 and DistilBERT (MobileBERT is discussed below). The main result is that the Hybrid Pruning model is as fast as the baseline and approaches the same accuracy while at the same time producing significantly smaller models in terms of density. Moving to the Hybrid Filled model leads to a further gain in speed at a small cost in model density. For instance, for the same F1 performance of 87.5, Hybrid Filled models display a 2.5x speedup against 1.88 for TinyBERT. TinyBERT and DistilBERT have 50% of BERT's encoder parameters, whereas Hybrid Filled models have 25% BERT parameters for the same level of accuracy.

The figures also include two intrinsic baselines: our reimplementation of Movement pruning and pure Block pruning. We find that our implementation of Movement pruning is highly effective at producing sparse models (even leading to a small increase in accuracy) but does not produce significant speedups. Square Block pruning does better, but not as well as hybrid blocks.

Table 2 gives a full comparison of models with different compression rates. As linear layers represent a very large part of the flops of a transformer model, this compression rate is actually a good measure of the maximum achievable speedup. This number is much higher than the actually measured speedup. This indicates that our setup for measur-

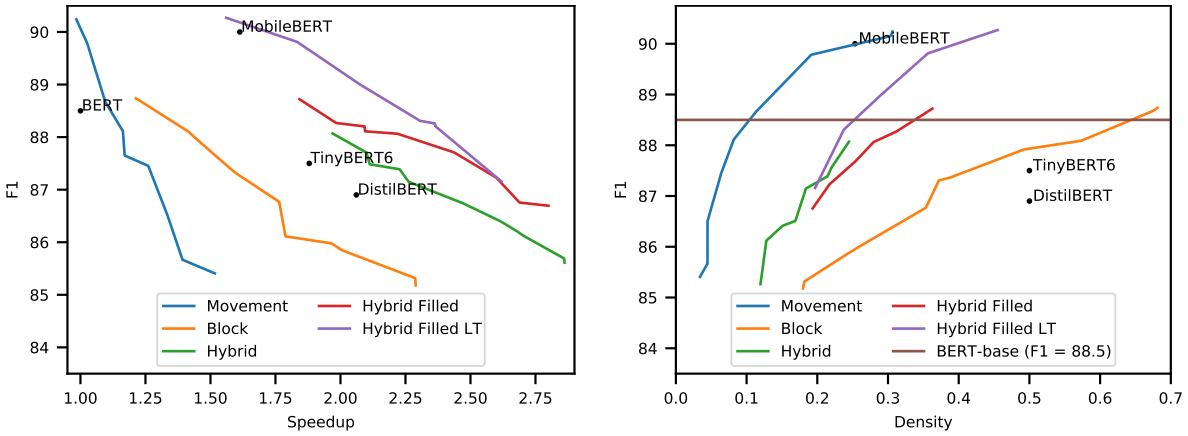

Figure 2: Comparison on SQuAD v1.1 of model F1 against speed and density (BERT-base reference). For each pruning method the pruned model is BERT-base, but different regularization values give different final sparsity levels. This translates into a tradeoff curve between accuracy and speedup specific to the method. Distilled networks (Mobile|Tiny|Distil)BERT are given as references. The higher the curve, the most accurate the model is for a given speed.

Model	Size	Compr.	Speed	F1	EM
BERT	85M	1.00	1.00	88.5	80.8
TinyBERT	42M	2.00	1.88	87.5	79.7
DistilBERT	42M	2.00	2.06	86.9	79.1
Hybrid Filled	30M	2.76	1.84	88.7	81.7
Hybrid Filled	21M	3.94	2.25	87.7	80.2
Hybrid	20M	4.09	1.97	88.07	80.6
Hybrid Filled	16M	5.17	2.69	86.8	78.8
Hybrid	15M	5.44	2.26	87.15	79.4
Hybrid	14M	5.91	2.24	86.51	78.7
Struct	14M	6.25	3.36	85.9	78.0
Hybrid	12M	6.63	2.61	86.41	78.3
Hybrid	10M	7.82	2.70	86.12	77.8
MobileBERT	21M	4.00	1.61	90.0	82.9
Hybrid Filled LT	38M	2.20	1.56	90.3	83.7
Hybrid Filled LT	20M	4.21	2.31	88.3	81.2
Hybrid Filled LT	16M	5.08	2.62	87.2	79.5

Table 2: SQuAD v1.1 pruned models with base/large teacher: Encoder Linear Parameters Count, Compression rate and Speed gain relative to BERT-base, F1 and Exact Match. LT stands for Large Teacher.

ing speed may underestimate the actual speedup one could obtain with those pruned models with specialized implementations. Hybrid Filled reaches a 2.25x speedup under minimal loss in accuracy. Struct pruning targeting MHA blocks directly can be even faster but leads to a stronger degradation in accuracy.

Table 3 shows the comparison between TinyBERT and a Hybrid pruned model of the same speed on several others tasks. Hybrid Pruning performs better on SQuAD v1.1, and approaches TinyBERT performance on other tasks.

Model	SQuAD v1.1 F1/EM	MNLI Acc (m/mm)	QQP F1	SST-2 Acc
TinyBERT	87.5/79.7	**84.6**/83.2	**88.0**	**93.0**
Hybrid	**88.1/80.6**	83.2/**83.6**	87.9	91.2

Table 3: Hybrid pruning/TinyBERT cross-task performance comparison. Speed is *at least* TinyBERT speed (1.88x BERT-base) for all networks and significantly sparser.

Comparison with MobileBERT All methods can be improved further using a larger teacher model. For these experiments, we compare with MobileBERT, which uses a BERT-large teacher and reaches an F1 of 90.0 on SQuAD v1.1 on its fastest version. It should be noted that MobileBERT makes use of additional optimizations not present in the original BERT-large we are using: LayerNorms are replaced by purely linear NoNorms, and GeLUs are replaced by ReLUs. For these experiments, we use a BERT-large teacher to perform meaningful comparisons, using our best method Hybrid Filled.

Figure 2 shows that we have comparable results on SQuAD v1.1, with a simpler optimization approach: we get a slightly better model (F1=90.3) for the same speedup of 1.6x, and we get a speedup of 2.2x at BERT-base accuracy (F1=88.5). We observe that using a large teacher is beneficial even at high levels of pruning: up to 80% of sparsity, the resulting student model has better accuracy for the

same number of parameters when using a BERT-large teacher instead of a base one. This trend reverses after this point: a larger teacher is detrimental to accuracy when the student is very heavily pruned.

Encoder-Decoder Finally, we apply these methods to two encoder-decoder architectures, BART-base and BART-large for the task of summarization. For these architectures, the decoder parameters are responsible for a majority of the computational costs, so these are our main focus. Voita et al. (2019) observed that for machine translation models, encoder heads were much easier to prune than decoder ones. We found similar results, e.g. for identical $\lambda_{\mathbf{att}}$ and $\lambda_{\mathbf{ffn}}$, the encoder was systematically more pruned than the decoder, for both MHA and FFN sub-layers. In order to increase speedup gain, we applied twice as much weight on the decoder compression, which resulted in even pruning ratios among the encoder and decoder.

Table 4 shows the main results. We see that Hybrid pruning leads to large decoder compression ratios (3.4 on BART-base and 3.5 BART-large) with only a small drop in ROUGE score. Speedups reach 1.4 times of the original speed. (Given the large decoder compression rates, we would expect larger speedups to be possible with further engineering of the inference.)

There is less comparable work for pre-trained encoder-decoders. We compare our approach with a distillation-based approach dBART (Shleifer and Rush, 2020). This approach yields a similar speedup gain with a smaller drop in performance but less sparsity. For models of comparable sizes (158M for our Hybrid NT vs 176M for dBART-6-6), we observe a drop of 0.7 in R2 and 0.4 in RL against 0.9 in R2 and 1.3 in RL for dBART-6-6. As with encoder-only models, the two approaches could likely be combined to yield even faster, smaller models.[3]

7 Analysis

Large Model Pruning To test that this approach scales to large models, we apply Hybrid pruning on BERT-large on SQuAD v1.1. We observe similar results: a 18% dense BERT-large has a F1 of 90.2, with a speedup of 3.2x compared to BERT-large with a F1 of 93.2. This pruned model is actually faster than a BERT-base model (Table 5). We can compare Hybrid Pruning of SQuAD v2 BERT-large models with the results of the structured pruning method described in McCarley (2019). For a 17% dense model, we obtain a F1 of 82.6, whereas structured pruning gets a 25% dense model with a F1 of 81.5.

This result is in line with Li et al. (2020): the larger the model, the more pruning is effective. When pruning a larger model, the final model is actually better than a smaller one with the same absolute number of parameters.

Block Size Influence Figure 3 shows the impact of different block sizes on Block pruning: pruning is done on attention layers and FFNs with the same square block size, from $(4, 4)$ to $(32, 32)$, with a BERT-base teacher. We can see that we reach the

Model	Size	DCp	Speed	R1	R2	RL
BART-large	353M	1.0	1.00	44.8	21.7	41.8
Hybrid NT	158M	2.0	-	44.3	21.0	41.4
Hybrid NT	108M	2.8	1.38	43.5	20.3	40.6
Hybrid NT	82M	3.5	1.39	42.7	19.6	39.9
BART-large†	353M	1.0	1.00	-	21.1	40.9
dBART-12-6†	252M	2.0	1.44	-	21.2	41.0
dBART-6-6†	176M	2.0	1.46	-	20.2	39.6
dBART-12-3†	201M	4.0	1.66	-	20.6	40.3
BART-base	99M	1.0	1.00	43.4	20.4	40.4
Hybrid NT	35M	2.6	1.19	42.2	19.4	39.2
Hybrid NT	23M	3.4	1.35	41.4	18.7	38.4

Table 4: BART pruned models fine-tuned on CNN: encoder-decoder linear parameters count, Decoder compression rate (DCp) and Speed gain (one forward prediction) relative to BART-large/base, dev ROUGE scores. † denotes test ROUGE scores taken from Shleifer and Rush (2020). NT stands for No Teacher.

Model	Size	Compr.	Speed	F1	EM
SQuAD v1.1					
BERT-large	227M	1.00	1.0	93.2	86.9
BERT-base	85M	2.66	3.1	88.5	80.8
Hybrid	54M	4.16	2.9	91.0	84.6
Hybrid	41M	5.59	3.2	90.2	83.7
SQuAD v2					
BERT-large	227M	1.00	1.0	85.8	82.8
StructuredQA	57M	4.00	–	81.5	–
Hybrid	38M	5.88	–	82.6	79.7

Table 5: BERT-large SQuAD pruned models. Reference speed is BERT-large.

[3]Distillation methods for text generation require generating pseudo-labels, a different process which is significantly slower than BERT distillation.

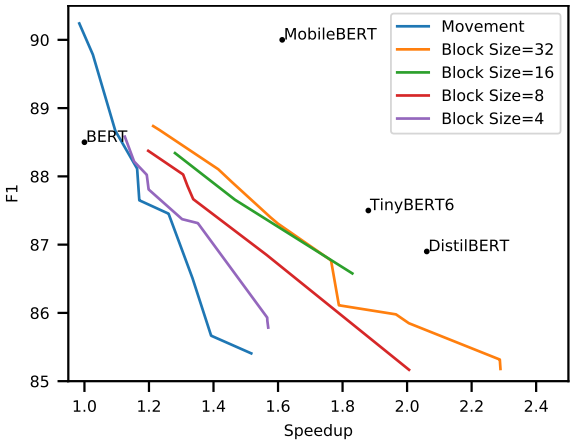

Figure 3: SQuAD v1.1 with Block Pruning: Influence of block size on F1.

BERT-base original F1 for all block sizes from 4 to 32, but with a speedup that increases with the block size. The maximum reachable speedup without F1 drop is 1.3 for a block size of 32. But when some drop of F1 is allowed, the speedup increases quickly with the block size and plateau when reaching 16. We then reach a speedup of 1.75 for an F1 drop of 2% and a block size of 32.

We also note that, with the original Movement Pruning method, we see some speedup due to full dimension pruning. This likely comes from our improved set of hyper-parameters (compared to the original paper), allowing us to remove some empty rows and columns in the FFN layers. However we see that using blocks leads to a significant speed improvement compared to Movement Pruning.

Quantization Quantization is often of critical importance for practical applications. We, therefore, wanted to check that our networks could be subjected to quantization without significant loss of accuracy, especially when considering the issues that could arise with the high level of sparsity of some FFNs. Table 6 shows the results of full 8-bit quantization tests on our models. These indicate that the method is compatible with quantization, and the models using quantization on top of our pruning method achieve very high gains in terms of size (as well as speed).

Impact of Distillation We report experimental results with the addition of a teacher distillation step as previous work showed this boosts movement pruning at little cost. In this section, we conduct an ablation study to evaluate the impact of distillation using a BERT-base teacher.

Model	Compress	EM	F1
BERT-base	1.0	80.8	88.5
Hybrid	3.3	80.2	87.8
+ quantization	13.3	77.8	86.3

Table 6: Results of pruned BERT-base on SQuAD v1.1 dev Exact Match and F1 score with 8-bit quantization.

Dataset	Size	Hybrid	Hybrid NT
QQP	24M	87.61	87.17
	21M	87.14	87.00
	10M	86.82	86.27
SST-2	70M	93.23	92.20
	42M	91.97	90.71
	18M	90.60	89.79

Table 7: Distillation ablation study of BERT-base on QQP and SST-2 dev of a BERT-base teacher. F1 and accuracy scores reported for QQP and SST-2 respectively. NT stands for No Teacher.

As shown in Table 7, combining hybrid pruning with distillation always performs better than pruning alone, but that it is not critical for the approach to work. The distillation effect is larger for smaller datasets such as SST-2, which are prone to over-fitting. We believe that the regularization brought by pruning and distillation counters over-fitting caused by the additional number of steps needed for pruning.

8 Conclusion

We have shown that we can extract small pruned models that are at an equivalent or better than distilled networks. This approach can be done during fine-tuning and not pre-training. The method does not resort to techniques such as data augmentation or architecture search, and it works on a diverse set of tasks and base models. As better and larger models are published at an increasing pace, we can rely on a simple and robust method to accelerate them on specific tasks without sacrificing accuracy and distribute these models easily while keeping most of the original model accuracy.

9 Impact

We expect the method presented here to contribute to the reduction of the compute resources and energy needed to perform natural language tasks, while preserving the original model performance. It will contribute additionally to alleviating privacy

concerns: smaller models running on user devices instead of server-side allow more information to stay private. This is especially relevant when considering the large anticipated demand for such NLP applications in the near future.

9.1 Acknowledgements

The authors would like to thank the anonymous reviewers, the Hugging Face team for the support, Nvidia for providing us some hardware for evaluation, and finally the open-source community for the numerous tools which made this research possible.

References

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432.

Zihan Chen, Hongbo Zhang, Xiaoji Zhang, and Leqi Zhao. 2018. Quora question pairs. *University of Waterloo*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635.

Mitchell A. Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing BERT: studying the effects of weight pruning on transfer learning. *CoRR*, abs/2002.08307.

Demi Guo, Alexander M. Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *CoRR*, abs/2012.07463.

Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1693–1701.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling BERT for natural language understanding. *CoRR*, abs/1909.10351.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hervé Jégou, and Tomás Mikolov. 2016. Fasttext.zip: Compressing text classification models. *CoRR*, abs/1612.03651.

Young Jin Kim and Hany Hassan Awadalla. 2020. Fastformers: Highly efficient transformer models for natural language understanding.

Yann LeCun, John S. Denker, and Sara A. Solla. 1989. Optimal brain damage. In *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pages 598–605. Morgan Kaufmann.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics.

Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joseph E. Gonzalez. 2020. Train large, then compress: Rethinking model size for efficient training and inference of transformers. *CoRR*, abs/2002.11794.

Christos Louizos, Max Welling, and Diederik P. Kingma. 2018. Learning sparse neural networks through l_0 regularization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Yihuan Mao, Yujing Wang, Chufan Wu, Chen Zhang, Yang Wang, Yaming Yang, Quanlu Zhang, Yunhai Tong, and Jing Bai. 2020. Ladabert: Lightweight adaptation of BERT through hybrid model compression. *CoRR*, abs/2004.04124.

J. S. McCarley. 2019. Structured pruning of a bert-based question answering model. *CoRR*, abs/1910.06360.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *CoRR*, abs/1905.10650.

Kenton Murray and David Chiang. 2015. Auto-sizing neural networks: With applications to n-gram language models. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 908–916. The Association for Computational Linguistics.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pages 784–789. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2383–2392. The Association for Computational Linguistics.

Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. 2019. What's hidden in a randomly weighted neural network? *CoRR*, abs/1911.13299.

Corby Rosset. 2020. Turing-nlg: A 17-billion-parameter language model by microsoft. *Microsoft Research Blog*, 2:13.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2020. Poor man's BERT: smaller and faster transformer models. *CoRR*, abs/2004.03844.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Victor Sanh, Thomas Wolf, and Alexander M. Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *CoRR*, abs/2005.07683.

Abigail See, Minh-Thang Luong, and Christopher D. Manning. 2016. Compression of neural machine translation models via pruning. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, pages 291–301. ACL.

Sam Shleifer and Alexander M. Rush. 2020. Pre-trained summarization distillation. *CoRR*, abs/2010.13002.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1631–1642. ACL.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. 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 2019, Hong Kong, China, November 3-7, 2019*, pages 4322–4331. Association for Computational Linguistics.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic BERT for resource-limited devices. *CoRR*, abs/2004.02984.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. 2019. Distilling task-specific knowledge from BERT into simple neural networks. *CoRR*, abs/1903.12136.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: The impact of student initialization on knowledge distillation. *CoRR*, abs/1908.08962.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *CoRR*, abs/1905.09418.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2019. Structured pruning of large language models. *CoRR*, abs/1910.04732.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

A Reproducibility & Hyper-Parameters

Code

The complete code to run the experiments, analyze the results and finally create the figures and tables in this paper is available on the Hugging Face nn_pruning repository, at https://github.com/huggingface/nn_pruning.

Hyperparameters

The hyperparameters of the experiments are available as JSON files (one file per task) in the same repository: each entry contains all the information to fine-tune and prune the model, its evaluation results, and detailed statistics about its final sparsity.

For example, the SQuAD V1 checkpoints referenced in this paper are listed with the hyperparameters and related information.

Checkpoints

Some of the models we produced during this research can be used directly from the Hugging Face model hub.

The other models and the checkpoints, including the intermediary ones that were saved during training, are available on Amazon S3.

B Additional Data

Block Shape & Head pruning

We show here the effect of the pattern on the head number reduction: using block instead of row/column pruning leads to a much larger number of pruned heads while improving accuracy, here on the SST-2 task.

We are using Block Movement pruning for each model, with different block patterns, pruning only the attention layers. Compression measures the reduction of the number of non-zero parameters in attention linear layers, whereas head compression measures the reduction of the number of complete non-zero heads.

Pattern	Compr.	Heads	Head Compr.	Accur.
BERT base	1x	144	1x	92.7
Rows/Cols	8.6x	86	1.7x	90.6
Block 32	4.7x	54	2.7x	91.1
Block 64	3.5x	51	2.8x	92.0

Table 8: Head pruning method comparison on SST-2

Pruning Methods Comparison

We select speed as our main metric to compare with other techniques, as it is the major practical measure of inference efficiency. On this metric, we decided to compare our models to the best models available i.e. the distilled models (MobileBERT, TinyBERT), even though the method is different, as they are the strongest "speed/accuracy" baseline available.

In Table 9 we compare Wang et al. (2019) with TinyBERT (Jiao et al., 2019) and MobileBERT (Sun et al., 2020).

Method	Speed	SST2	MRPC	STS-B	QNLI
Wang et al.	<1.5x	92.09	88.61	88.18	89.05
TinyBERT	2x	93.1	87.3	83.7	90.4
MobileBERT	4x	92.8	88.8	84.4	90.6

Table 9: Distillation/Structured Pruning Comparison

We compare as well to Hybrid pruning, with and without a teacher, with the unstructured methods from Sanh et al. (2020) (the original Movement Pruning method we are using) and Gordon et al. (2020), and with Sajjad et al. (2020) (dropping full layers), in Table 10.

Model	Spd	Cp	MNLI (m/mm)	QQP F1/Acc	SST-2 Acc
BERT base	1	1	84.5/85.1	88.1/91.1	92.7
Mvmt NT	~ 1	10	80.7/81.1	87.1/90.5	-
Hybrid NT	3.5	10	79.4/79.9	86.0/89.3	87.0
Mvmt	~ 1	10	81.2/81.8	86.8/90.2	-
Hybrid	3.5	10	80.4/81.1	86.4/89.8	89.7
Hybrid NT	3	4.5	81.7/81.8	87.0/90.3	89.8
Hybrid	3	4.5	82.7/82.8	87.4/90.6	90.6
Sajjad	< 2	2	81.1/ -	- /90.4	90.3
Gordon	-	2	82.6/-	- /90.3	90.8
Hybrid NT	1.6	2	83.2/83.3	87.2/90.4	90.7
Hybrid	1.6	2	83.7/84.1	88.3/91.3	92.0

Table 10: Pruning Methods Comparison. Mvmt is Movement Pruning Sanh et al. (2020), Sajjad is Sajjad et al. (2020), Gordon is Gordon et al. (2020). NT is for No Teacher. Spd is speed multiplier, Cp is for parameters compression rate.