Blockwise Self-Attention for Long Document Understanding

Jiezhong Qiu[1*], **Hao Ma**[2], **Omer Levy**[2], **Wen-tau Yih**[2], **Sinong Wang**[2], **Jie Tang**[1]
[1]Department of Computer Science and Technology, Tsinghua University
[2]Facebook AI
`qiujz16@mails.tsinghua.edu.cn`
`{haom,omerlevy,scottyih,sinongwang}@fb.com`
`jietang@tsinghua.edu.cn`

Abstract

We present BlockBERT, a lightweight and efficient BERT model for better modeling long-distance dependencies. Our model extends BERT by introducing sparse block structures into the attention matrix to reduce both memory consumption and training/inference time, which also enables attention heads to capture either short- or long-range contextual information. We conduct experiments on language model pre-training and several benchmark question answering datasets with various paragraph lengths. BlockBERT uses 18.7-36.1% less memory and 12.0-25.1% less time to learn the model. During testing, BlockBERT saves 27.8% inference time, while having comparable and sometimes better prediction accuracy, compared to an advanced BERT-based model, RoBERTa.

1 Introduction

Recent emergence of the *pre-training* and *fine-tuning* paradigm, exemplified by methods like ELMo (Peters et al., 2018), GPT-2/3 (Radford et al., 2019; Brown et al., 2020), BERT (Devlin et al., 2019), XLNet (Yang et al., 2019), RoBERTa (Liu et al., 2019) and ALBERT (Lan et al., 2019), has drastically reshaped the landscape of the natural language processing research. These methods first pre-train a deep model with language model objectives using a large corpus and then fine-tune the model using in-domain supervised data for target applications. Despite its conceptual simplicity, this paradigm has re-established the new state-of-the-art baselines across various tasks, such as question answering (Devlin et al., 2019), coreference resolution (Joshi et al., 2019b), relation extraction (Soares et al., 2019) and text retrieval (Lee et al., 2019; Nogueira and Cho, 2019), to name a few.

Building such models in practice, however, is an extremely resource-intensive process. For instance, the training of BERT-family models is notoriously expensive. Devlin et al. (2019) report that it takes four days to pre-train BERT-Base/BERT-Large on 4/16 Cloud TPUs. In order to reduce the pre-training time of RoBERTa to 1 day, Liu et al. (2019) use 1,024 V100 GPUs. One crucial factor contributing to the long training time is the memory consumption of these deep models, as it directly affects the batch size. Although the fine-tuning stage is relatively inexpensive, the memory issue still restricts the scenarios in which BERT can be used. For instance, "it is currently not possible to re-produce most of the BERT-Large results on the paper using a GPU with 12GB-16GB of RAM, because the maximum batch size that can fit in memory is too small.[1]"

Although one may think that model size is the main contributor to the large memory consumption, our analysis (Section 2.1) shows that one of the main bottlenecks is actually dot-product self-attention, operated in multiple layers of Transformers (Vaswani et al., 2017), the building block of BERT. As the attention operation is quadratic to the sequence length, this fundamentally limits the maximum length of the input sequence, and thus restricts the model capacity in terms of capturing long-distance dependencies. As a result, downstream tasks have to either truncate their sequences to leading tokens (Nogueira and Cho, 2019) or split their sequences with a sliding window (Joshi et al., 2019a,b). Ad-hoc handling of long sequences is also required in the pre-training stage, such as updating the model using only short sequences in the early stage (Devlin et al., 2019).

Common strategies for reducing memory consumption, unfortunately, do not work. For instance,

[*]This work was done when the first author was an intern at Facebook AI. Code is available at `https://github.com/xptree/BlockBERT`

[1]`github.com/google-research/bert`

shrinking the model by lowering the number of layers L, attention heads A, or hidden units H leads to significant performance degradation (Vaswani et al., 2017; Devlin et al., 2019) and does not address the long sequence issue. Alternatively, general low-memory training techniques, such as microbatching (Huang et al., 2018) and gradient checkpointing (Chen et al., 2016) essentially trade off training time for memory consumption, prolongs the already lengthy training process.

In this work, we explore a different strategy, *sparsifying the attention layers*, intending to design a lightweight and effective BERT that can model long sequences in a memory-efficient way. Our BlockBERT extends BERT by introducing sparse block substructures into attention matrices to reduce both memory consumption and the number of floating-point operations (FLOPs), which also enables attention heads to capture either short- or long-range contextual information. Compared to the previous method that also enforces sparsity (Child et al., 2019), our approach is much simpler mathematically and very easy to implement. More importantly, the results of experiments conducted on several benchmark question answering datasets with various paragraph lengths show that BlockBERT performs comparably or even better than the original BERT-family models, while enjoying an 18.7-36.1% reduction in memory usage, a 12.0-25.1% reduction in training time, and a 27.8% reduction in inference time.

The rest of the paper is organized as follows. Section 2 gives a brief introduction of the BERT model, along with an in-depth analysis of its memory usage during training time. We describe our proposed model in Section 3 and contrast it with existing methods that aim for creating a lighter model. Section 4 presents the experimental results and ablation studies, followed by a survey of other related work in Section 5 and the conclusion in Section 6.

2 Background: Memory Bottleneck in Training BERT

We briefly review BERT and introduce its memory profiling in this section. Following the paradigm of language model pre-training and down-stream task fine-tuning, BERT (Devlin et al., 2019) consists of multiple layers of bidirectional Transformers (Vaswani et al., 2017), where each Transformer encoder has a multi-head self-attention layer and a position-wise feed-forward layer. Using the same notation as in (Devlin et al., 2019), we denote the number of Transformer layers by L, the number of hidden units by H, the number of attention heads by A, the sequence length by N, and the batch size by B. We also assume the feed-forward hidden unit size to be $4H$.[2]

2.1 Memory Profiling

Training BERT is a memory-intensive process. In order to identify the bottleneck, we follow the memory model proposed by Sohoni et al. (2019), where memory usage throughout neural network training is categorized into three main types: (1) **Model memory** is used to store model parameters; (2) **Optimizer memory** is the additional memory used by the specific learning algorithm during the process; (3) **Activation memory** consists of the outputs of each layer, which are cached for reuse in backpropagation to compute gradients.

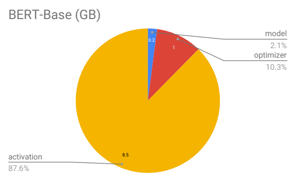
Take BERT-Base training as an example. The model has 110 million parameters, so model memory occupies 0.2 GB if parameters are stored in half-precision floating-point format (FP16). For Adam (Kingma and Ba, 2014), the optimizer needs additional memory to store the gradients, first moments, and second moments of model parameters. If stored using the same precision, the optimizer memory should be three times of model memory.[3] To calculate the exact size of activation memory is not trivial because it depends heavily on the implementation of the toolkit. Instead, we measure it empirically by training BERT-Base using Adam with a memory profiler (more details are provided in Appendix A.2).

We use 32 NVIDIA V100 GPUs for training. Every single GPU thus consumes a minibatch of size $b = B/32 = 8$. Figure 1(a) shows the profiling result for a single GPU, where the model/optimizer/activation memory consumes 0.21/1.03/8.49 GB, respectively. We can see that activation memory accounts for the vast majority of the total GPU memory (87.6%) and is thus the bottleneck. Notice that although our analysis is done on BERT-Base, it can also be generalized to BERT-Large and other models such as RoBERTa (Liu et al., 2019), XLNet (Yang et al., 2019) and AL-

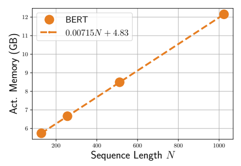
[2]The default parameter settings for BERT-Base and BERT-Large can be found in Appendix A.1

[3]In the current PyTorch Adam implementation, the first and second moments are stored in single precision. Consequently, BERT's optimizer memory (1 GB) is five times of model memory (0.2 GB).

BERT (Lan et al., 2019).



(a) BERT-Base Training Memory Profiling



(b) Regression Analysis on Activation Memory

Figure 1: Memory Profiling for BERT.

2.2 A Regression Analysis on Activation Memory

For BERT, or more specifically, Transformer, the activation memory corresponds to intermediate results of different layers. It grows linearly in all the model hyper-parameters, except the sequence length N, due to the attention layers. To quantify the linear and quadratic components in the activation memory more clearly, we conduct a regression analysis as follows. Assume that the activation memory (in each GPU) is a polynomial $a_2bN^2 + a_1bN + a_0$, where b is the batch size in each GPU and a_i $(i = 0, 1, 2)$ are coefficients to be determined. If we fix the total number of tokens in a GPU to be constant (in our case, we fix $b \times N = 4096$), we should have a linear function w.r.t. N, i.e., $4096a_2N + 4096a_1 + a_0$. We enumerate N from $\{128, 256, 512, 1024\}$ in our experiments, and plot the corresponding profiled activation memory in Figure 1(b). Using ordinary least squares (OLS), with $b \times N = 4096$, the estimated linear function for activation memory is $0.00715 \times N + 4.83$, where the first term corresponds to the $O(N^2)$ component. When $N = 512$ (i.e., $b = 8$), we can see that for BERT-Base, the $O(N^2)$ component accounts for 3.66 GB, and the $O(N)$ component accounts for 4.83 GB. When the sequence length N increases to 1024 (i.e., $b = 4$), the $O(N^2)$ component increases to 7.32 GB, while the $O(N)$ part is unchanged.

2.3 Techniques for Reducing Traing Memory

Observing that activation memory is the training bottleneck, we discuss common memory reduction techniques below.

Low Precision (Micikevicius et al., 2017) Low precision is to use half-precision/mixed-precision for training neural networks. This technique has been widely used in Transformer training (Ott et al., 2019; Liu et al., 2019). In this work, we already assume to use mixed-precision training by default, as indicated in the aforementioned analysis.

Microbatching (Huang et al., 2018) Microbatching is to split a batch into small microbatches (which can be fit into memory), and then run forward and backward passes on them separately with gradients for each micro-batch accumulated. Because it runs forward/backward pass multiple times for a single batch, it trades off time for memory.

Gradient Checkpointing (Chen et al., 2016) Gradient checkpointing saves memory by only caching activations of a subset of layers. The un-cached activations will be recomputed during backpropagation from the latest checkpoint. This strategy trades off time for memory by repeating computations and will obviously extend training time.

Knowledge Distillation (Hinton et al., 2015) Knowledge distillation aims to compress and transfer knowledge from a teacher model to a simpler student model. However, knowledge distillation relies on a teacher model (which is still expensive in training time) and usually suffers from a certain degree of performance degradation.

As common techniques are limited in reducing both the training time and memory usage, we investigate how to optimize the dot-product attention layers and introduce our approach next.

3 Model: BlockBERT

Following (Vaswani et al., 2017), the dot-product attention in Transformer is defined as:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V},$$

where $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N \times d}$ with N to be the sequence length and d to be a hidden dimension. As we can see, the inner product between $\boldsymbol{Q}$ and $\boldsymbol{K}$ consumes $O(N^2)$ memory. One simple way to reduce the memory consumption of attention is to sparsify the attention matrix. Suppose we have a masking matrix $\boldsymbol{M} \in \{0, 1\}^{N \times N}$, we define a masked version of attention as follows:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{M}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}} \odot \boldsymbol{M}\right)\boldsymbol{V}, \tag{1}$$

with operator $\odot$ defined by

$$(\boldsymbol{A} \odot \boldsymbol{M})_{ij} = \begin{cases} \boldsymbol{A}_{ij} & \text{if } \boldsymbol{M}_{ij} = 1 \\ -\infty & \text{if } \boldsymbol{M}_{ij} = 0 \end{cases}.$$

In this work, we design $\boldsymbol{M}$ to be a *sparse block matrix*, which not only reduces memory and the number of floating-point operations (FLOPs) but also benefits from efficient dense matrix support from deep learning frameworks, such as PyTorch and Tensorflow. More formally, we split the length-N input sequence into n blocks, with each block of length $\frac{N}{n}$.[4] The $N \times N$ attention matrix is then partitioned into $n \times n$ blocks, where each block matrix is of the size $\frac{N}{n} \times \frac{N}{n}$. We define a sparse block matrix $\boldsymbol{M}$ by a permutation π of $\{1, 2, \cdots, n\}$:

$$\boldsymbol{M}_{ij} = \begin{cases} 1 & \text{if } \pi\left(\lfloor \frac{(i-1)n}{N} + 1 \rfloor\right) = \lfloor \frac{(j-1)n}{N} + 1 \rfloor, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

By writing $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ as block matrices, such that $\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_1^\top & \cdots & \boldsymbol{Q}_n^\top \end{bmatrix}^\top, \boldsymbol{K} = \begin{bmatrix} \boldsymbol{K}_1^\top & \cdots & \boldsymbol{K}_n^\top \end{bmatrix}^\top$ and $\boldsymbol{V} = \begin{bmatrix} \boldsymbol{V}_1^\top & \cdots & \boldsymbol{V}_n^\top \end{bmatrix}^\top$ and pluging them into Equation 1, we can formally define Blockwise Attention as follows:

$$\begin{aligned} &\text{Blockwise-Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{M}) \\ &= \begin{bmatrix} \text{softmax}\left(\frac{\boldsymbol{Q}_1 \boldsymbol{K}_{\pi(1)}^\top}{\sqrt{d}}\right) \boldsymbol{V}_{\pi(1)} \\ \vdots \\ \text{softmax}\left(\frac{\boldsymbol{Q}_n \boldsymbol{K}_{\pi(n)}^\top}{\sqrt{d}}\right) \boldsymbol{V}_{\pi(n)} \end{bmatrix}. \end{aligned} \quad (3)$$

Equation 3 only needs to compute and store $\boldsymbol{Q}_i \boldsymbol{K}_{\pi(i)}^\top$ $(i = 1, \cdots n)$, each has size $\frac{N}{n} \times \frac{N}{n}$. In other words, BlockBERT reduces both $O(N^2)$ memory consumption and FLOPs by a factor of n, since $\frac{N}{n} \times \frac{N}{n} \times n = \frac{N \times N}{n}$.

3.1 Blockwise Multi-Head Attention

Analogous to Multi-head Attention (Vaswani et al., 2017), we allow queries, keys, and values to be projected multiple times and perform blockwise attentions in parallel. Moreover, different blockwise attention heads can use different masking matrices. The outputs of multiple heads are then concatenated and aggregated with another linear projection. Let A be the number of attention heads and H the number of hidden units. *Blockwise multi-head attention* is formally defined as follows:

$$\begin{aligned} &\text{Blockwise-Multi-head-Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) \\ &\qquad = \text{Concat}(\text{head}_1, \cdots \text{head}_A) \boldsymbol{W}^O, \end{aligned}$$

where for each head i, $i = 1, 2, \cdots, A$,

$$\text{head}_i = \text{Blockwise-Attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V, \boldsymbol{M}_i),$$

[4] We assume N can be divided by n. If not, we pad the input sequence to make N divisible.

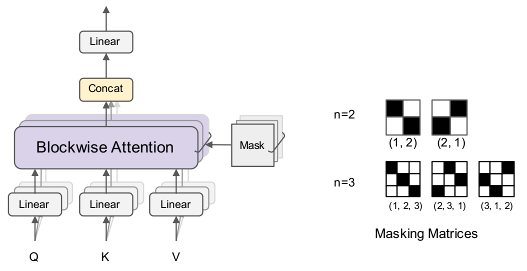


Figure 2: Architecture of Blockwise Multi-head Attention, which acts as building blocks of BlockBERT. The key idea is to introduce a sparse block masking matrix to the $N \times N$ attention matrix. The right panel shows the masking matrices we use when $n = 2, 3$. For $n = 2$, the masking matrices are defined by permutation $(1, 2)$, $(2, 1)$ and have 50% non-zeros. For $n = 3$, the masking matrices are defined by permutation $(1, 2, 3)$, $(2, 3, 1)$, and $(3, 1, 2)$ and have 33.33% non-zeros.

with $d = \frac{H}{A}, \boldsymbol{W}_i^Q, \boldsymbol{W}_i^K, \boldsymbol{W}_i^V \in \mathbb{R}^{H \times d}$ and the projection matrix $\boldsymbol{W}^O \in \mathbb{R}^{H \times H}$. Each masking matrix $\boldsymbol{M}_i$ is determined by a permutation π_i according to Equation 2. In particular, we choose π from permutations generated by *shifting one position*: $\sigma = (2, 3, \cdots, n, 1)$, i.e., we select $\pi \in \{\sigma, \sigma^2, \cdots, \sigma^n\}$. For example, with 12 attention heads ($A = 12$) and 2 blocks ($n = 2$), we can assign 10 heads to permutation $(1, 2)$ and the other 2 heads to permutation $(2, 1)$. Figure 2 illustrates the blockwise multi-head attention with block number $n \in \{2, 3\}$. Blockwise sparsity captures both local and long-distance dependencies in a memory-efficiency way, which is crucial for long-document understanding tasks. For instance, the identity permutation, i.e., $(1, 2, \cdots, n)$, enables each token to attend to its nearby tokens in self-attention, while other permutations allow tokens within the same block attending to tokens in another block. Our proposed BlockBERT essentially replaces the multi-head attention layers in Transformer/BERT with blockwise multi-head attention.

3.2 Analysis of Memory Usage Reduction

To validate our claim that BlockBERT with $n \times n$ blocks can reduce the $O(N^2)$ memory usage by a factor of n, we perform the same memory profiling as described in sections 2.1 and 2.2. Again, We fix the number of tokens in each GPU ($b \times N = 4096$) and choose N from $\{128, 256, 512, 1024, 2048\}$.[5] As we can see from Figure 3 and Table 1, the empirical results align well with the theoretical values.

[5] We use GPUs of 16 GB memory for profiling. BERT with $N = 2048$ fails due to an out-of-memory error.

When we set the number of blocks to be 2 and 3 for BlockBERT, the estimated $O(N^2)$ activation memory decreases to 1/2 and 1/3 of BERT's $O(N^2)$ activation memory, respectively. As shown in Table 2, for the sequence length $N = 512$, BlockBERT with 2 and 3 blocks saves 18.7% and 23.8% overall memory, respectively. The saving is more significant for longer sequences. When $N = 1024$, the overall memory reduction of BlockBERT with 2 and 3 blocks is 27.3% and 36.1%, respectively.

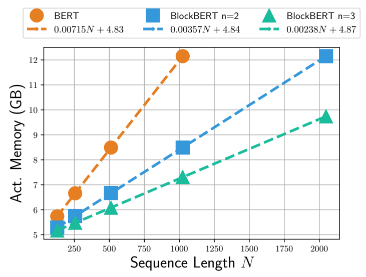


Figure 3: Regression analysis on activation memory for BERT and BlockBERT.

Table 1: Estimated $O(N^2)$ and $O(N)$ activation memory for BERT and BlockBERT.

			Act. Mem. (GB)	
N	b	Model	$O(N)$	$O(N^2)$
512	8	BERT	4.83	3.66
		BlockBERT n=2	4.84	1.83
		BlockBERT n=3	4.87	1.22
1024	4	BERT	4.83	7.32
		BlockBERT n=2	4.84	3.66
		BlockBERT n=3	4.87	2.44

4 Experiments

We evaluate the pre-training and fine-tuning performance of BlockBERT. In particular, when $n = 2$, we denote 10:2 to be the configuration which assigns 10 heads to permutation $(1, 2)$ and 2 to permutation $(2, 1)$; when $n = 3$, we denote 8:2:2 to be the configuration which assigns 8, 2, 2 heads to permutation $(1, 2, 3)$, $(2, 3, 1)$, and $(3, 1, 2)$, respectively. We compare BlockBERT with the following baselines:

Google BERT Google BERT is the official pre-trained model from (Devlin et al., 2019).

RoBERTa-2seq & RoBERTa-1seq We compare with two versions of RoBERTa (Liu et al., 2019). RoBERTa-2seq is trained with both masked language model (MLM) task and next sentence prediction (NSP) task, while RoBERTa-1seq refers to the pre-training model with only the MLM task.

SparseBERT We pre-train BERT models with its Transformer encoder replaced by a Sparse Transformer encoder (Child et al., 2019). We set its sparsity hyper-parameters stride $\ell = 128$ and expressivity $c = 32$.[6] The attention masking matrix used in Sparse Transformer and more implementation details are discussed in Appendix A.3. A similar architecture was adopted in GPT-3 (Brown et al., 2020).

4.1 Pre-training

All the models follow the BERT-Base setting, i.e., $L = 12, H = 768, A = 12$, and are trained on the same corpus — BooksCorpus and English Wikipedia with uncased word piece tokens. Thus all models use the same vocabulary as Google BERT (uncased version) with vocabulary size 30,522. We fix the number of tokens per batch $B \times N = 131,072$, i.e., if sequence length $N = 512$ then batch size $B = 256$, if sequence length $N = 1024$ then batch size $B = 128$. The detailed pre-training configuration is listed in Appendix A.1. Moreover, the pre-training of SparseBERT and BlockBERT follows the RoBERTa-1seq setting, i.e., we drop the NSP (Next Sentence Prediction) task, and an input sequence is up to N tokens until it reaches a document boundary.

A summary of the pre-training performance comparison between BlockBERT and RoBERTa-1seq is shown in Table 2. Besides memory saving, we also achieve a significant speedup. For example, when $N = 1024$, BlockBERT ($n = 2$) reduces the training time from RoBERTa's 9.7 days to 7.5 days.

4.2 Fine-tuning Tasks

We evaluate BlockBERT on several question answering tasks, including SQuAD 1.1/2.0 (Rajpurkar et al., 2018) and five other tasks from the MrQA shared task[7] — HotpotQA (Yang et al., 2018), NewsQA (Trischler et al., 2017), SearchQA (Dunn et al., 2017), TriviaQA (Joshi et al., 2017) and NaturalQA (Kwiatkowski et al., 2019). Since MrQA does not have an official test set, we follow Joshi et al. (2019a) to split the devel-

[6] We adopt Sparse Transformer implemented by Fairseq, which first computes the $N \times N$ attention matrix, and then masks it to be a sparse one. This implementation cannot avoid the $O(N^2)$ attention computation, and thus has a similar training time/memory cost to RoBERTa.

[7] mrqa.github.io

Table 2: Pre-training Performance Analysis.

N	Model	Training Time (day)	Memory (per GPU, GB)	Heads Config.	Valid. ppl
512	RoBERTa-1seq	6.62	9.73	-	3.58
	BlockBERT n=2	5.83 (-12.0%)	7.91 (-18.7%)	10:2	3.56
	BlockBERT n=3	5.80 (-12.5%)	7.32 (-23.8%)	8:2:2	3.71
1024	RoBERTa-1seq	9.66	13.39	-	3.60
	BlockBERT n=2	7.51 (-22.3%)	9.73 (-27.3%)	9:3	3.57
	BlockBERT n=3	7.23 (-25.1%)	8.55 (-36.1%)	8:2:2	3.63

opment set evenly to build a new development set and test set.

These QA datasets have different paragraph length distributions and are thus ideal for testing the effectiveness of BlockBERT[8]. For example, SQuAD, NaturalQA, and HotpotQA consist of mostly short paragraphs (shorter than 512), while paragraphs in SearchQA (average length 1,004) and TriviaQA (average length 934) have around 1,000 tokens. When the input sequence is longer than N, we follow the common practice (Joshi et al., 2019a) to split it using a sliding window of size N and stride 128. This means that for SearchQA and TriviaQA, a model with $N = 512$ can only capture half of the context, while a model with $N = 1024$ can accept the whole paragraph as input.

For all models, we adopt the same fine-tuning QA setup from Devlin et al. (2019). The tokenized paragraph $(p_1, \cdots, p_s)$ and question $(q_1, \cdots, q_t)$ are concatenated to be a sequence `[CLS]`$q_1 \cdots q_t$`[SEP]`$p_1 \cdots p_s$`[SEP]`. The sequence is then fed into the pre-trained model with two extra linear layers for predicting the start and end positions of the answer spans. The detailed fine-tuning setting is listed in Appendix A.4. Table 3 and Table 4 report the experimental results.

BlockBERT (n=2) v.s. RoBERTa-1seq Comparing BlockBERT with RoBERTa-1seq in the case of $N = 512$, we observe an absolute F1 difference from 0.04 (in NaturalQA) to 1.18 (in NewsQA), with an average of 0.55. For $N = 1024$, BlockBERT achieves more comparable or even better performance to RoBERTa-1seq, In SearchQA, NewsQA and HotpotQA, BlockBERT achieves absolute F1 improvement of 0.39, 0.44 and 0.23, respectively.

BlockBERT v.s. SparseBERT For $N = 512$, it is interesting that BlockBERT with 3 blocks (density 33.33%) performs better then SparseBERT (density 44.20%) in both SQuAD and MrQA tasks. Similar results can be observed for $N = 1024$, too. These results show that off-diagonal masking matrices, e.g., the masking matrix defined by permutation $(2, 3, 1)$ and $(3, 1, 2)$, play crucial roles in BlockBERT. Furthermore, BlockBERT with 2 blocks achieve a more significant improvement.

Table 3: Dev set results on SQuAD 1.1/2.0. The result of XLNet(-Base) is from Yang et al. (2019).

N	Model	SQuAD 1.1 EM	SQuAD 1.1 F1	SQuAD 2.0 EM	SQuAD 2.0 F1
-	Human Perf.	82.30	91.20	86.80	89.40
512	Google BERT	81.19	88.45	74.08	77.16
	XLNet	-	-	78.46	81.33
	RoBERTa-2seq	82.91	89.78	75.79	79.17
	RoBERTa-1seq	**84.43**	**91.48**	**79.22**	**82.27**
	SparseBERT	80.49	88.09	74.15	76.96
	BlockBERT n=2, 10:2	*84.08*	*90.77*	*78.34*	*81.46*
	BlockBERT n=3, 8:2:2	82.37	89.64	77.33	80.33
1024	RoBERTa-1seq	**84.58**	**91.14**	**79.34**	**82.26**
	SparseBERT	81.02	88.37	74.51	77.57
	BlockBERT n=2, 9:3	*83.65*	*90.74*	*78.55*	*81.45*
	BlockBERT n=3, 8:2:2	82.74	90.05	76.79	79.84

Effect of Long Sequence Pre-training Our observations are twofold: (1) Long sequence pre-training benefits long sequence fine-tuning. In TriviaQA and SearchQA, of which paragraph lengths are around 1024, pre-training models with $N = 1024$ achieve significantly better performance. (2) The heterogeneity of pre-training and fine-tuning sequence length may hurt performance. For example, in SQuAD, we do not see significant performance gain by using pre-trained models with $N = 1024$; in HotpotQA and NewsQA, longer sequence pre-training even hurts performance.

Effect of #Blocks It is not surprising that BlockBERT with 2 blocks ($n = 2$) performs bet-

[8]The detailed paragraph length distributions can be found in Appendix A.5

ter than that with 3 blocks ($n = 3$), because it keeps more attention matrix entries. The biggest difference is in SQuAD 2.0 and NewsQA with $N = 1024$, where we observe an absolute loss of 1.6 F1 by increasing block number from 2 to 3.

Efficient inference with BlockBERT We benchmark test efficiency of RoBERTa and BlockBERT. The benchmark code follows huggingface[9]. All experiments are run 30 times on a 32GB V100 GPU with half precision (FP16). We report the average running time in Table 5. As we can see, BlockBERT does achieve speedup and memory reduction during test time. Take 8×1024, i.e., batch size $B = 8$, sequence length $N = 1024$, as an example, we can see that BlockBERT with 2 blocks saves 27.8% of test time, and BlockBERT with 3 blocks saves more (30.4%). As for memory, we can observe that RoBERTa cannot handle an input of size 16×1024, while it is possible for BlockBERT to work on it.

In summary, not only BlockBERT saves training/inference time and memory, but it also has a competitive and sometimes better performance, especially for tasks with longer sequences. This demonstrates the effectiveness of our blockwise multi-head attention approach.

4.3 Ablation Study

We fix the assignment of attention heads in the above experiments. For example, BlockBERT with sequence length $N = 512$ and 2 blocks is trained with ten heads using permutation $(1, 2)$ and the other two using permutation $(2, 1)$. However, there are other ways to assign twelve attention heads, e.g., seven heads for permutation $(1, 2)$ and the other five for permutation $(2, 1)$. It would be interesting to see how the assignment of heads affects model performance. In this section, we grid search attention head assignments and plot their best validation performance in 1.2M training steps. The results are shown in Figure 4.

Our observations are threefold: (1) Identity permutations, i.e., $(1, 2)$ and $(1, 2, 3)$, are important. As shown in Figure 4, all optimal solutions assign considerable attention heads to block-diagonal matrices, since those matrices enable each token to attend to its nearby tokens; (2) Non-identity permutations follow the rule of "vital few and trivial many." Although identity permutations are important, assigning all attention heads to them (corresponding to 12:0 and 12:0:0 in Figure 4) significantly hurts performance, since the model can not learn long-term dependencies with only identity permutation; (3) Pre-training performance and fine-tuning performance are correlated but not always consistent. When $n = 3$, pre-training performance suggests 10:1:1 to be the best head assignment — ten heads for permutation $(1, 2, 3)$, one head for $(2, 3, 1)$ and one head for $(3, 1, 2)$, but we observe that the configuration of 8:2:2 achieves better performance in fine-tuning tasks.

5 Related Work

In this section, we review the related work of memory optimization for neural network training and recent efforts to simplify Transformer and BERT.

5.1 Low-memory neural networks training

Due to the large size of model parameters and deep architectures, modern neural networks training requires significant amounts of computing resources. As a result, there is an increasing interest in training neural networks with low memory (Sohoni et al., 2019). Mainstream techniques mostly address this problem with a better system or engineering design, such as low-precision training (Micikevicius et al., 2017), microbatching (Huang et al., 2018) and gradient checkpointing (Chen et al., 2016). Alternatively, there also exists some research focusing on the theoretical aspect, including the recently proposed lottery ticket hypothesis (Frankle and Carbin, 2018).

5.2 Efficient Transformer

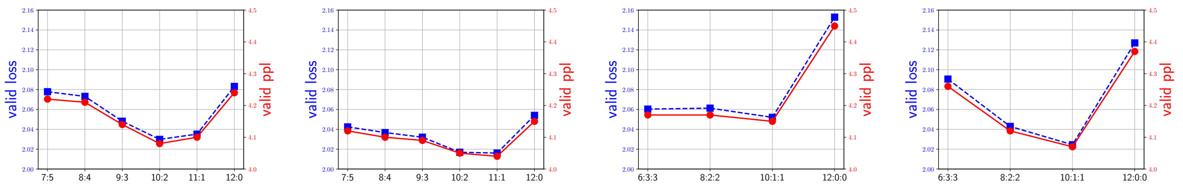
Since the invention of Transformer (Vaswani et al., 2017) and its successful application to masked language model pre-training (Devlin et al., 2019; Radford et al., 2019; Yang et al., 2019; Liu et al., 2019; Lan et al., 2019), several approaches have been proposed to simplify the model and its training process. We summarize these attempts as follows:

Attention layer simplification There are currently two lines of research trying to simplify the multi-head attention layers. The first one focuses on attention matrix sparsification. Notable examples include Star Transformer (Guo et al., 2019), Sparse Transformer (Child et al., 2019), Adaptive Sparse Transformer (Correia et al., 2019; Sukhbaatar et al., 2019), Log-Sparse Transformer (Li et al., 2019) , Reformer (Kitaev et al.,

[9] `github.com/huggingface/transformers/blob/master/examples/benchmarks.py`

Table 4: MrQA test results (Tasks are sorted decreasingly by average paragraph length).

N	Model	SearchQA EM	SearchQA F1	TriviaQA EM	TriviaQA F1	NewsQA EM	NewsQA F1	NaturalQA EM	NaturalQA F1	HotpotQA EM	HotpotQA F1
512	Google BERT	74.94	80.37	70.18	75.35	51.27	66.25	66.13	78.29	60.50	77.08
	RoBERTa-2seq	76.12	81.74	71.92	76.79	52.45	66.73	66.98	78.63	61.52	77.81
	RoBERTa-1seq	**77.09**	**82.62**	**73.65**	**78.22**	**56.13**	**70.64**	**67.14**	**79.07**	**62.77**	**79.28**
	SparseBERT	73.36	79.01	68.71	73.15	51.18	65.47	65.53	77.46	58.54	74.85
	BlockBERT n=2, 10:2	*76.68*	*82.33*	*72.36*	*77.53*	*54.66*	*69.46*	*66.94*	*79.03*	*62.13*	*79.15*
	BlockBERT n=3, 8:2:2	75.54	81.07	72.05	76.74	53.82	68.39	66.14	78.47	60.64	77.46
1024	RoBERTa-1seq	77.47	83.12	**75.29**	**80.20**	55.00	69.64	**68.28**	**80.35**	61.89	78.71
	SparseBERT	74.83	80.54	70.56	75.34	51.67	67.16	65.07	77.31	59.65	76.02
	BlockBERT n=2, 9:3	***77.95***	***83.51***	*75.06*	*79.41*	***55.44***	***70.08***	*67.31*	*79.39*	***62.13***	***78.94***
	BlockBERT n=3, 8:2:2	76.98	82.76	74.78	79.28	53.48	68.50	65.91	78.20	61.89	78.18



(a) $N = 512, n = 2$ (b) $N = 1024, n = 2$ (c) $N = 512, n = 3$ (d) $N = 1024, n = 3$

Figure 4: Ablation over blockwise attention heads assignment.

Table 5: Test time statistics (sec) for different input size. OOM indicates out-of-memory.

$B \times N$	8×1024	16×1024	24×1024	32×1024
RoBERTa	0.1371	OOM	OOM	OOM
BlockBERT n=2	0.0990	0.1869	OOM	OOM
BlockBERT n=3	0.0954	0.1790	0.2634	OOM

2020) and Longformer (Beltagy et al., 2020). However, due to the insufficient support for sparse tensors from the current deep learning platforms, some of them have to represent a sparse matrix using a dense matrix with a binary mask or rely on customized CUDA kernels (Gray et al., 2017). As a result, the speed-up or reduction in memory consumption is sometimes limited in practice. The second line of research prunes redundant attention heads. Examples include (Voita et al., 2019) and (Michel et al., 2019). Our BlockBERT model belongs to the first category, as we sparsify the attention matrix by replacing it with a block sparse matrix.

Reducing model size for pre-training Knowledge distillation (Hinton et al., 2015) is a general technique that aims to compress and transfer knowledge from a teacher model to a simpler student model. There are two recent efforts that apply knowledge distillation to BERT pre-training for reducing model size: TinyBERT (Jiao et al., 2019) distills BERT using a smaller Transformer, and Tang et al. (2019) distills BERT with a BiLSTM (Hochreiter and Schmidhuber, 1997). In contrast, ALBERT (Lan et al., 2019) is a notable work that does not take the knowledge distillation approach. It uses parameter-sharing to reduce the number of parameters of the BERT model. As discussed in section 2.1, parameter-sharing reduces both model memory and optimizer memory. These two parts account for about 12.4% of total training memory for BERT-base. As for efficiency, parameter-sharing reduces communication complexity in distributed training and thus saves training time as well.

In the aforementioned efficient Transformers, the model quality is often demonstrated by comparable language model perplexity, or equivalently the bits per word/byte. It is often implicitly assumed that similar language model perplexity implies similar pre-training model quality, namely the same performance on the downstream tasks. We would like to point out that this assumption does not necessarily hold. For example, the experiments on the Enwik8 dataset by Child et al. (2019) demonstrates that Sparse Transformer "surpasses the 1.03 state-of-the-art (bits per byte) for a similarly-sized Transformer-XL and matching the 0.99 (bits per byte) of a model trained with more than double the number of parameters". However, if we compare SparseBERT (pre-training model with Sparse Transformer backbone) against XLNet (Yang et al., 2019) (pre-training model with Transformer-XL backbone) in SQuAD, Table 3 shows that XLNet still outperforms SparseBERT significantly. Therefore, we believe that it is necessary to conduct a comprehensive study and evaluation of existing efficient Transformer models when used for masked language model pre-training. Limited by resources, in this work, we mainly compare BlockBERT to pre-training using Sparse Transformer (Child et al., 2019), which is the earliest attempt to design efficient Transformer models and also the key contributor to the success of GPT-3 (Brown et al., 2020). We plan to benchmark more models in the future.

6 Conclusion

In this work, we study the lightweight BERT model with the goal of achieving both efficiency and effectiveness. We profile and analyze the memory bottlenecks of BERT and focus on optimize dot-product self-attention, which consumes quadratic memory with respect to the sequence length. To reduce both time and memory consumption, we present BlockBERT, which sparsifies the attention matrices to be sparse block matrices. The proposed model achieves time and memory saving without significant loss of performance.

In the future, we plan to benchmark more efficient Transfomers in language model pre-training and fine-tuning. We also would like to explore more applications of BlockBERT on NLP tasks involving long sequences such as coreference resolution (Joshi et al., 2019b) and document-level machine translation (Miculicich et al., 2018), and also non-NLP tasks such as protein sequence modeling (Rives et al., 2019; Rao et al., 2019).

Acknowledgments

The authors would like to thank Zhilin Yang, Danqi Chen, Yinhan Liu, Mandar Joshi and Luke Zettlemoyer for the helpful suggestions.

References

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Gonçalo M Correia, Vlad Niculae, and André FT Martins. 2019. Adaptively sparse transformers. *arXiv preprint arXiv:1909.00015*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT' 2019*, pages 4171–4186.

Matthew Dunn, Levent Sagun, Mike Higgins, V Ugur Guney, Volkan Cirik, and Kyunghyun Cho. 2017. Searchqa: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*.

Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Scott Gray, Alec Radford, and Diederik P Kingma. 2017. Gpu kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224*.

Qipeng Guo, Xipeng Qiu, Pengfei Liu, Yunfan Shao, Xiangyang Xue, and Zheng Zhang. 2019. Star-transformer. In *NAACL-HLT' 2019*, pages 1315–1325.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, and Zhifeng Chen. 2018. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*.

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. *arXiv preprint arXiv:1909.10351*.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2019a. Spanbert: Improving pre-training by representing and predicting spans. *arXiv preprint arXiv:1907.10529*.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL' 17*, pages 1601–1611.

Mandar Joshi, Omer Levy, Daniel S Weld, and Luke Zettlemoyer. 2019b. Bert for coreference resolution: Baselines and analysis. *arXiv preprint arXiv:1908.09091*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300*.

Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. 2019. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *arXiv preprint arXiv:1907.00235*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *arXiv preprint arXiv:1905.10650*.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. 2017. Mixed precision training. *arXiv preprint arXiv:1710.03740*.

Lesly Miculicich, Dhananjay Ram, Nikolaos Pappas, and James Henderson. 2018. Document-level neural machine translation with hierarchical attention networks. In *EMNLP' 18*, pages 2947–2954.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*.

Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Peter Chen, John Canny, Pieter Abbeel, and Yun Song. 2019. Evaluating protein transfer learning with tape. In *Advances in Neural Information Processing Systems*, pages 9686–9698.

Alexander Rives, Siddharth Goyal, Joshua Meier, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, and Rob Fergus. 2019. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*, page 622803.

Livio Baldini Soares, Nicholas FitzGerald, Jeffrey Ling, and Tom Kwiatkowski. 2019. Matching the blanks: Distributional similarity for relation learning. *arXiv preprint arXiv:1906.03158*.

Nimit Sharad Sohoni, Christopher Richard Aberger, Megan Leszczynski, Jian Zhang, and Christopher Ré. 2019. Low-memory neural network training: A technical report. *arXiv preprint arXiv:1904.10631*.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive attention span in transformers. *arXiv preprint arXiv:1905.07799*.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. 2019. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2017. Newsqa: A machine comprehension dataset. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 191–200.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

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. *arXiv preprint arXiv:1905.09418*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *EMNLP' 18*.

A Appendix

A.1 Notations and Pre-training Hyper-parameters

The notations and pre-training hyper-parameters are listed in Table 6 and Table 7.

Table 6: BERT notations.

	Description	Base	Large
B	Batch size	256	256
A	# Self-attention heads	12	16
L	# Layers	12	24
H	# Hidden units	768	1024
$4H$	# Feed-forward hidden units	3072	4096
N	Sequence length	512	512

A.2 Profiler Implementation

Among the three types of training memory, model memory and optimizer memory is relatively easy to profile (can be computed by enumerating each tenor and summing up `tensor.numel() * tensor.element_size()`). To calculate activation memory, (Sohoni et al., 2019) traverse PyTorch's autograd graph and sum up the necessary storage space. They find that the summation of model memory, optimizer memory, and activation memory matches PyTorch memory profiling tool[10].

[10] `torch.cuda.max_memory_allocated`

Table 7: Pre-training hyper-parameters.

Hyper-parameter	Value
Vocabulary Size	30,522
Dropout	0.1
Attention dropout	0.1
Warmup steps	10K
Weight decay	0.01
Max steps	2.4M
Initial learning rate	0.00025
Learning rate decay	Linear
Adam ϵ	1e-8
Adam β_1	0.9
Adam β_2	0.999
Gradient Clipping	1.0

Based on their observation, we use the following quantity as an estimate to activation memory

$$\text{max memory allocated} - \text{model memory} - \text{optimizer memory} \tag{4}$$

When profiling BERT, we first pre-train it for 1000 steps, and then compute its model and optimizer memory. Finally, we estimate its activation memory according to Equation 4.

A.3 SparseBERT

The sparse masking matrices we use for Sparse Transformer (Child et al., 2019) are shown in Figure 5. We adopt the implementation of Sparse Transformer from Fairseq[11]. The Fariseq version is implemented in a direct way, with the goal of comparing performance, not speed. We first compute the N^2 attention matrix and then mask it to be a sparse matrix according to the sparse pattern defined in Sparse Transformer paper. Consequently, this implementation of SparseBERT has very close training time/memory cost as RoBERTa (as it can not avoid the $O(N^2)$ attention computation). We did so because the code released by Sparse Transformer is based on Tensorflow and relies on customized CUDA kernels, but our pre-training is done using PyTorch.

A.4 Fine-tuning Settings

Our fine-tuning is implemented based on code base from HuggingFace[12] and SpanBERT (Joshi et al., 2019a). We use `max_sequence_length=`N, i.e., we allow fine-tuning task to input sequences as long as the pre-training model. If the input sequence is too long to fit the

[11] `github.com/pytorch/fairseq/blob/master/fairseq/modules/sparse_multihead_attention.py`.

[12] `github.com/huggingface/pytorch-transformers`

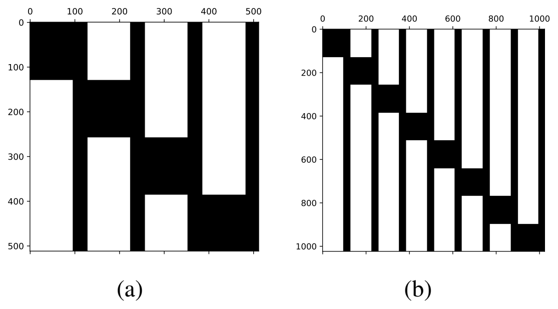


Figure 5: The sparse masking matrices we use in Sparse Transformer (fixed mode) encoder. White color indicates attention values to be masked. (a) $N = 512, \ell = 128, c = 32$, density 44.20%; (b) $N = 1024, \ell = 128, c = 32$, density 34.97%.

`max_sequence_length`=N constraints, we use a sliding window of stride 128 to split it. We grid search learning rate from {5e-6, 1e-5, 2e-5, 3e-5, 5e-5} and batch size from {16, 32}. The fine-tuning is performed for 4 epoches.

A.5 Paragraph-Length Distribution

The paragraph-length distribution of SQuAD and MrQA datasets is shown in Figure 6.

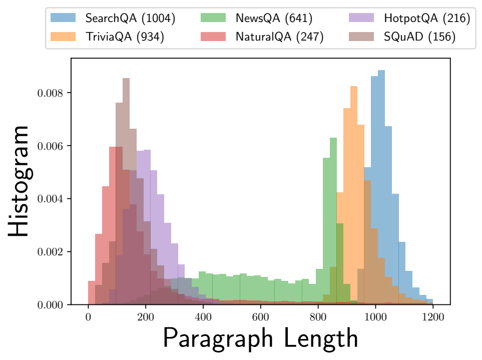


Figure 6: Paragraph-length (after tokenization) distribution. The distribution of SQuAD 2.0 is very similar to SQuAD 1.1, so we only plot SQuAD 1.1 here.