FASTER AND JUST AS ACCURATE: A SIMPLE DECOMPOSITION FOR TRANSFORMER MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large pre-trained Transformers such as BERT have been tremendously effective for many NLP tasks. However, inference in these large capacity models is prohibitively slow and expensive. Transformers are essentially a stack of self-attention layers which encode each input position using the entire input sequence as its context. However, we find that it may not be necessary to apply this expensive sequence-wide self-attention over at all layers. Based on this observation, we propose a decomposition to a pre-trained Transformer that allows the lower layers to process segments of the input independently enabling parallelism and caching. We show that the information loss due to this decomposition can be recovered in the upper layers with auxiliary supervision during fine-tuning. We evaluate decomposition with pre-trained BERT models on five different paired-input tasks in question answering, sentence similarity, and natural language inference. Results show that decomposition enables faster inference (up to 4x), significant memory reduction (up to 70%), while retaining most (up to 99%) of the original performance. We will release the code at <anonymized url>.

1 INTRODUCTION

Inference in large Transformer-based NLP models such as BERT (Devlin et al., 2019) requires prohibitively high-levels of compute, making it expensive to support large volume processing in data centers, and almost infeasible to run on resource constrained mobile devices. These Transformer models create effective representations using self-attention, a mechanism that allows them to effectively account for wide textual contexts. However, applying self-attention over the *entire* input for *all* layers is computationally expensive. This raises a natural question: *Is self-attention over the entire input necessary in all of the layers?*

Previous studies (Tenney et al., 2019; Hao et al., 2019; Clark et al., 2019b) have shown that lower layers tend to capture syntactic phenomena that mostly depend on local contexts and that higher layers capture more semantic phenomena that are relevant to downstream tasks, which depend on longer global contexts. This suggests that considering only local context in lower layers of Transformer and considering full global context in upper layers can provide speedup at a very small cost in terms of effectiveness.

In this work we focus on paired-input NLP tasks such as reading comprehension, natural language inference and sentence pair similarity. These tasks provide a natural boundary for the locality of text (e.g., question vs. passage in QA). Because of this natural decomposition in two segments, we can compute representations for lower layers with only the local segment as the context and compute representations for upper layers with both segments as the context. This decomposition technique has multiple benefits: It allows for parallel processing of each segment, caching of segments that are available offline, and a significant reduction in runtime memory. Moreover, since the architecture remains largely same, the original pre-trained weights can be reused in the decomposed model. To compensate for the differences in the decomposed setting, we augment the fine-tuning loss on the target task with a distillation loss that minimizes the output-level as well as layer-level divergences.

We evaluate the decomposition idea using the BERT model on five different pairwise tasks. The decomposition achieves substantial speedup (2 to 4.3x) and reduction in memory (51.1% to 76.8%) for only small loss in effectiveness (0.2 to 1.8 points). Moreover, we find that with decomposition

the larger BERT model can even run faster than the original smaller BERT model, while still being more accurate.

2 RELATED WORK

Speeding up inference in a model requires reducing the amount of compute involved. There are two broad related directions of prior work:

(i) **Compression techniques** can be used to reduce model size through low rank approximation (Zhang et al., 2015; Kim et al., 2015; Tai et al., 2015; Chen et al., 2018), and model weights pruning (Guo et al., 2016; Han et al., 2015), which have been shown to help speedup inference in CNN and RNN based models. For Transformers, Michel et al. (2019) explore pruning the attention heads to gain inference speedup. This is an orthogonal approach that can be combined with our decomposition idea. However, for the paired-input tasks we consider, pruning heads only provides limited speedup. In more recent work Ma et al. (2019) propose approximating the quadratic attention computation with a tensor decomposition based multi-linear attention model. However, it is not clear how this multi-linear approximation can be applied to pre-trained Transformers like BERT.

(ii) **Distillation techniques** can be used to train smaller student networks to speedup inference. Tang et al. (2019) show that BERT can be used to guide designing smaller models (such as single-layer BiLSTM) for multiple tasks. But for the tasks we study, such very small models suffer a significant performance drop. For instance there is a 13% accuracy degration on MNLI task. Another closely related recent work is DistillBERT (Sanh et al., 2019), which trains a smaller BERT model (half the size of BERT-base) that runs 1.5 times faster than the original BERT-base. However, the distilled model incurs a significant drop in accuracy. More importantly, this distillation model usually undergo expensive pre-training on the language modeling tasks before they can be fine-tuned for the downstream tasks.

In this work, we ask if can we speedup the inference of Transformer models without compressing or removing model parameters. Part of the massive success of pre-trained Transformer models for many NLP task is due to a large amount of parameters capacity to enable complex language representations. The decomposition we propose makes minimal changes retaining the overall capacity and structure of the original model but allows for faster inference by enabling parallel processing and caching of segments.

3 METHOD

Transformers process the entire input sequence through multiple layers of self-attention, each of which involves an expensive transformation that is quadratic in the number of tokens in the input sequence. In the case of paired-input tasks with query and candidate texts (e.g. question and passage in QA, premise and hypothesis in NLI), the Transformers compute attention over a concatenation of both texts. This results in highly effective representations of the input pair, but this comes at a high-cost in terms of time complexity. If you want to reduce complexity, one natural question to ask is whether we can decompose the Transformer function over segments of the input, trading some representational power for efficiency.

Consider the question-answering use case shown in Figure 1, where the standard input-wide self-attention implicitly models question self-attention, the passage self-attention, and the two question and passage cross-attentions. The cross-attentions allow the question and passage to influence each other and produce effective representations for the target task. However, if we can process the two segments separately, we can improve efficiency in multiple ways. We get a basic reduction in compute because we no longer perform cross-attention, going from $O(p+q)^2$ to $O(p^2+q^2)$. More im-

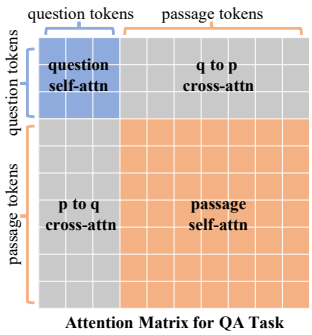


Figure 1: Illustration of the attention matrix components for QA task.

portantly we can get important gains through parallelism and caching. The question and passage self-attentions can be computed in parallel. Since the passage representation is no longer dependent on the question, they can be computed off-line and cached[1]. The trade-off, however, is that we loose some representation effectiveness because we no longer use information from the other segment. We argue that this is a good trade-off from two viewpoints:

(i) First, we can expect to achieve a good trade-off in models with multiple layers, where the cross-attention is less important in lower layers when compared to the upper layers. Figure 2 supports this argument using the similarity of the representations of BERT layers for a single passage when computed using five different questions. The variance is smaller for lower layers and increases for higher layers. This is also in agreement with results on probing tasks which suggest that lower layers tend to model mostly local phenomena (e.g., POS, syntactic categories), while higher layers tend to model more semantic phenomena that are task dependent (e.g, entity co-reference) relying on wider contexts. We further posit that in these multi-layer models information loss in the lower layers can be potentially compensated for by the higher layers.

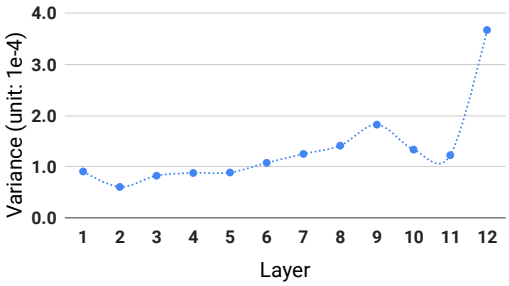


Figure 2: Average variance of passage representations when paired with different questions at different layers.

(ii) The approach requires minimal change to the overall Transformer architecture, which means we can reuse the pre-trained weights and expect that fine-tuning on the target task by itself will be sufficient to obtain high performance.

3.1 Decomposing a Transformer

First, let's define formally the computation that a Transformer undertakes for a paired-task containing two segments of text, T_a and T_b. Let the token embedding representations of segment T_a be $\mathbf{A} = [\mathbf{a_1}; \mathbf{a_2}; ...; \mathbf{a_q}]$ and of T_b be $\mathbf{B} = [\mathbf{b_1}; \mathbf{b_2}; ...; \mathbf{b_p}]$. The full input sequence $\mathbf{X}$ can be expressed by concatenating the token representations from segment T_a and T_b as $X = [A; B]$.

The Transformer encoder has n layers (denoted L_i for layer i), which transform this input sequentially: $X^{l+1} = L_i(X^l)$. For the details of the Transformer layer, we refer the reader to (Vaswani et al., 2017). We denote the application of a stack of layers from layer i to layer j be denoted as $L_{i:j}$.

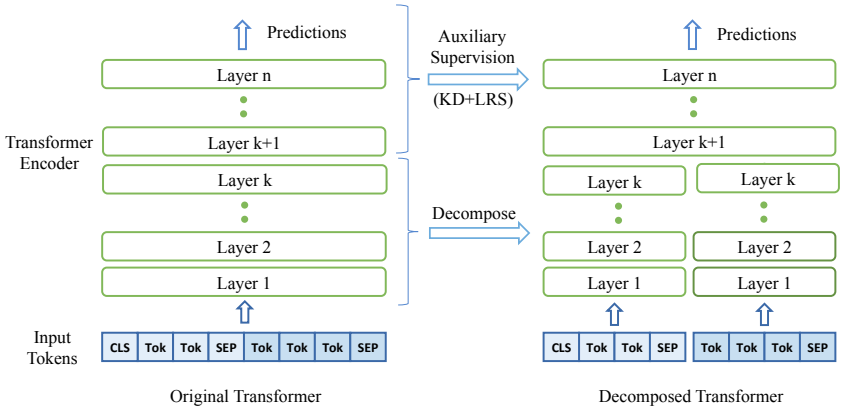


Figure 3: Decomposing Transformers up to layer k, which enables encoding each segment independently from layer 1 to layer k. Auxiliary supervision of upper layer information from the original model further helps the decomposed model to compensate for information loss in the lower layers. KD is Knowledge Distillation loss and LRS is Layerwise Representation Similarity loss.

[1]There is a storage cost associated with caching. We show in Section 5.2 that storage cost is many orders of magnitude smaller compared to inference costs.

Full Transformer: The output representations of the full Transformer, $\mathbf{A^n}$ and $\mathbf{B^n}$ can be expressed as:

$$[\mathbf{A^n}; \mathbf{B^n}] = L_{1:n}([\mathbf{A^0}; \mathbf{B^0}]) \tag{1}$$

Decomposed Transformer: Figure 3 shows a schematic of our model. We decompose the computation of lower layers (up to layer k) by simply removing the cross-interactions between T_a and T_b representations. Here k is a hyper-parameter. The output representations of the decomposed Transformer, $\mathbf{A^n}$ and $\mathbf{B^n}$ can be expressed as:

$$[\mathbf{A^n}; \mathbf{B^n}] = L_{k+1:n}([L_{1:k}(\mathbf{A^0}); L_{1:k}(\mathbf{B^0})) \tag{2}$$

3.2 Supervising Decomposed Transformer

The decomposed Transformer can be used in the same way as the original Transformer. Since the decomposed Transformer retains much of the original structure, we can initialize this model with the pre-trained weights of the original Transformer and fine-tune directly on downstream tasks. However, the decomposed Transformer looses some information in the representations of the lower layers. The upper layers can learn to compensate for this during fine-tuning. However, we can go further and use the original model behavior as an additional source of supervision.

Towards this end, we first initialize the parameters of decomposed Transformer with the parameters of a pre-trained full Transformer, and fine-tune it on the downstream tasks. To guide the learning of decomposed-Transformer further, we add auxiliary losses that makes predictions and the upper layer representations of decomposed Transformer closer to the predictions and corresponding layer representations of the full Transformer during fine-tuning.

Knowledge Distillation Loss: We use this loss to bring the prediction distribution of the decomposed Transformer closer to that of the full Transformer during fine-tuning. For this, we minimize the Kullback—Leibler divergence between decomposed Transformer prediction distribution P_A and full Transformer prediction distribution P_B:

$$\mathcal{L}_{kd} = D_{KL}(P_A \| P_B)$$

Layerwise Representation Similarity Loss: We use this loss to bring the layer representations of decomposed Transformer closer to those of full Transformer during fine-tuning. For this, we minimize the euclidean distance between token representations of the upper layers of decomposed Transformer and the full Transformer. Let $\mathbf{v_i}^j$ be the token representations from i^{th} layer (n layers) and j^{th} token (m tokens) of the full Transformer. Likewise, let $\mathbf{u_i}^j$ be the token representations from i^{th} layer (n layers) and j^{th} token (m tokens) of the decomposed Transformer. Finally, let decomposed Transformer be decomposed up to layer k. We compute the Layerwise Representation Similarity loss for fine-tuning decomposed Transformer as following:

$$\mathcal{L}_{lrs} = \sum_{i=k}^{n} \sum_{j=1}^{m} \|\mathbf{v_j^i} - \mathbf{u_j^i}\|^2$$

We add the Knowledge Distillation Loss and Layerwise Representation Similarity Loss along with the Task Specific Supervision Loss ($\mathcal{L}_{ts}$) and learn their relative importance via hyper-parameter tuning:

$$\mathcal{L}_{\text{total}} = \gamma\mathcal{L}_{ts} + \alpha\mathcal{L}_{kd} + \beta\mathcal{L}_{lrs} \tag{3}$$

We use the Bayesian Optimization (Močkus, 1975) to tune the γ, α and β instead of simple trial-and-error or grid/random search. The Bayesian process is designed to minimize the number of steps required to find a combination of hyper-parameters that are close to the optimal one.

4 EXPERIMENTS

4.1 DATASETS

We use the pre-trained uncased BERT base and large[2] models on five different *paired-input* problems covering 3 QA tasks, 1 natural language inference task and 1 sentence-level semantic similarity task:

SQuAD v1.1 (Stanford Question Answering Dataset) (Rajpurkar et al., 2016) is an extractive question answering datasets containing >100,000 question and answer pairs generated by crowd workers on Wikipedia articles.

RACE (Lai et al., 2017) is reading comprehension dataset collected from the English exams that are designed to evaluate the reading and reasoning ability of middle and high school Chinese students. It has over 28,000 passages and 100,000+ questions.

BoolQ (Clark et al., 2019a) consists of 15942 yes/no questions that are naturally occurring in unprompted and unconstrained settings.

MNLI (Multi-Genre Natural Language Inference) (Williams et al., 2018) is a crowd-sourced corpus of 433k sentence pairs annotated with textual entailment information.

QQP (Quora Question Pairs) (Iyer et al., 2019) consists of over 400,000 potential duplicate question pairs from Quora.

For all 5 tasks, we use the standard splits provided with the datasets but in addition divide the original training data further to obtain a 10% split to use for tuning hyper-parameters (tune split), and use the original development split for reporting efficiency (FLOPs, memory usage) and effectiveness metrics (accuracy or F1 depending on the task).

4.2 IMPLEMENTATION DETAILS

We implement all models in TensorFlow 1.14 (Abadi et al., 2015) based on the original BERT codebase (Devlin et al., 2019). We perform all experiments on one TPU v3-8 node (8 cores, 128GB memory) with *bfloat16* format enabled. We measure the FLOPs and memory consumption through the TensorFlow Profiler[3]. For decomposed Transformer models, we tune the hyperparameters for weighting different losses using bayesian optimizaiton libray (Nogueira, Fernando, 2019) with 50 iterations on the tune split (10% of the original training sets) and report the performance numbers on the original dev sets. The search range is [0.1, 2.0] for the 3 hyper-parameters.

For Decomposable BERT, we compute the representation for one of the input segments offline and cache it. For QA we cache the passages, for natural language inference, we cache the premise[4] and for question similarity we cache the first question[5].

4.3 RESULTS

Table 1 shows the main results comparing performance, inference speed and memory requirements of BERT-base and Decomposed BERT-base when using nine lower layers, and three upper layers (see Subsection 4.4 for the impact of the choice of upper/lower splits). We observe a substantial speedup and significant memory reduction in all the datasets, while retaining most of the original model's effectiveness (as much as 98.4% on SQuAD and 99.8% on QQP datasets). Efficiency improvements increase with the size of the text segment that can be cached.

Small Distilled or Large Decomposed? Table 2 compares performance, speed and memory of BERT-base, BERT-large and Decomposed BERT-large. Decomposed BERT-large is 1.6 times faster than the smaller BERT-base model. Decomposing the larger model turns out to be also more effec-

[2]Whole Word Masking version

[3]`https://www.tensorflow.org/api_docs/python/tf/profiler/profile`

[4]One use case is where we want to find (premise) sentences from a collection that support information contained in a query (hypothesis) sentence.

[5]One use case is FAQ retrieval, where a user question is compared against a collection of previously asked questions

	Avg. Input Tokens	BERT base	Decomp-BERT base	Performance Drop (absolute | %age)	Inference Speedup (times)	Memory Reduction (%age)
SQuAD	320	88.5	87.1	1.4 | 1.6	3.2x	70.3
RACE	2048	66.3	64.5	1.8 | 2.7	3.4x	72.9
BoolQ	320	77.8	76.8	1.0 | 1.3	3.5x	72.0
MNLI	120	84.4	82.6	1.8 | 2.1	2.2x	56.4
QQP	100	90.5	90.3	0.2 | 0.2	2.0x	50.0

Table 1: (i) Performance of BERT-base vs Decomp-BERT-base, (ii) Performance drop, inference speedup and inference memory reduction of Decomp-BERT-base over BERT-base for 5 tasks. Decomp-BERT-base uses nine lower layers, and three upper layers with caching enabled. For SQuAD and RACE we also train with the auxiliary losses, and for the others we use the main supervision loss – the settings that give the best effectiveness during training. Note that the choice of the loss doesn't affect the efficiency metrics.

	Performance (Squad-F1)	Speed (GFLOPs)	Memory (MB)
BERT-large	92.3	204.1	1549.6
BERT-base	88.5	58.4	584.2
Decomp-BERT-large	90.8	47.7	359.7

Table 2: Performance, Inference Speed and Memory for different models on SQuAD.

	Tesla V100 GPU	Intel i9-7900X CPU	OnePlus 6 Phone
BERT-base	0.22	5.90	10.20*
Decomp-BERT-base	0.07	1.66	3.28*

Table 3: Inference latency (in seconds) on SQuAD datasets for BERT-base vs Decomp-BERT-base, as an average measured in batch mode. On the GPU and CPU we use a batch size 32 and on the phone (marked by *) we use a batch size of 1.

tive than using the smaller base model (+2.3 points) This shows that with decomposition, a large Transformer can run faster than a smaller one which is half its size, while also being more accurate.

We note that distilling a larger model into a smaller can yield better accuracy than training a smaller model from scratch. As far as we know, there are two related but not fully comparable results. (1) Tang et al. (2019) distill BERT to a small LSTM based model where they achieve 15x speedup but at a significant drop in accuracy of more than 13 points on MNLI. (2) Sanh et al. (2019) distill BERT to a smaller six layer Transformer, which can provide 1.6x speedup but gives >2 points accuracy drop on MNLI and >3 points F1 drop on SQuAD. A fair comparison requires more careful experimentation exploring different distillation sizes which requires repeating pre-training or data augmentation – an expensive proposition.

Device Results: To evaluate the impact on different devices, we deployed the models on three different machines (a GPU, CPU, and a mobile phone). Table 3 shows the average latency in answering a question measured on a subset of the SQuAD dataset. On all devices, we get more than three times speedup.

4.4 Ablation Study

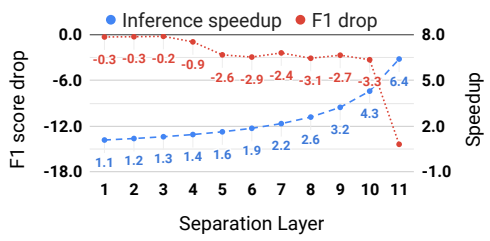
Table 4 shows the contribution of auxiliary losses for fine-tuning Decomposed-BERT on SQuAD dataset. The drop in effectiveness when not using Layerwise Representation Similarity (LRS in table), and Knowlege Distillation (KD) losses shows the utility of auxiliary supervision.

Figure 4a and figure 4b shows how the effectiveness and inference speed of Decomposed-BERT changes as we change the separation layer. The plot shows that inference speed up roughly scales quadratically with respect to the separation layer number. The drop in effectiveness, on the other hand, is negligible for separating at lower layers (until layer 3 for the base model and until layer 13

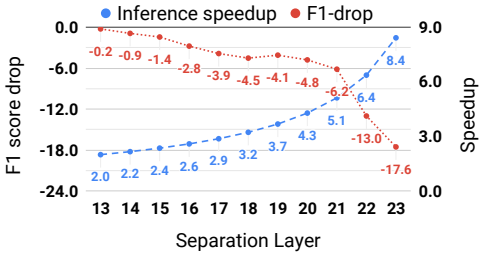
	Base Model	Large Model
BERT	88.5	92.3
Decomp-BERT	87.1	90.8
w/o LRS	86.2	88.9
w/o KD & LRS	85.8	87.5

Table 4: Ablation analysis on SQuAD datasets for Decomp-BERT-base and Decomp-BERT-large models.

for the large model) and increases slowly after that with a dramatic increase in the last layers closest to the output. The separation layer choice thus allows trading effectiveness for inference speed.



(a) F1 drop versus speedup on SQuAD for Decomposed BERT-base without auxiliary supervision.



(b) F1 drop versus speedup on SQuAD for Decomposed BERT-large without auxiliary supervision.

Figure 4: F1 drop versus speedup of Decomposed BERT model (without auxiliary supervision) when separating at different layers.

5 Analysis and Discussion

5.1 Understanding Transformer Lower and Upper Layers

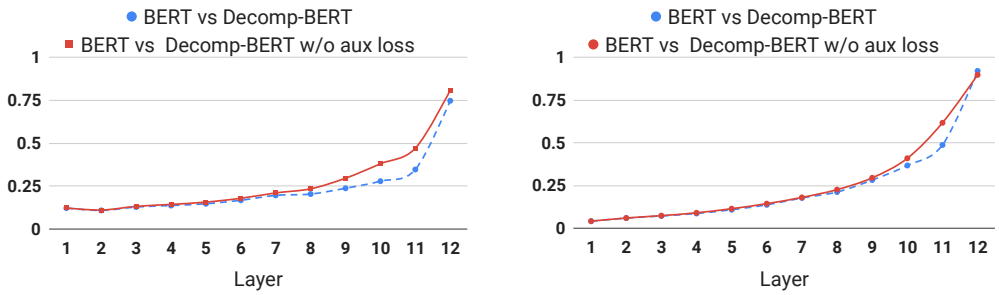
The main difference between the original BERT and the decomposed BERT is the absence of cross attention in the lower layers. We analyze the differences between the representations of the two models across all layers. To this end, we randomly select 100 passages from SQuAD dev dataset as well as randomly selecting 5 different questions that already exist in the dataset associated with each passage. For each passage, we encode all 5 question-passage pair sequence using both the fine-tuned original BERT-base model and the decomposed model, and compute their distance of the vector representations at each layer. Figure 5 shows the averaged distances of both the question and passage at different layers.

Figure 5a indicates that the lower layer representations of the passage for both models remain similar but the upper layer representations differ significantly, supporting the argument that lower layers tend to capture more of local context than global contexts. In addition, the figure also shows that using the auxiliary supervision of upper layers has the desired effect of forcing the decomposed BERT model to produce representations that are closer to the original model.

Figure 5b shows the distance of question representations between original BERT and decomposed BERT, which also supports the findings from the passage representation. One minor difference is the smaller gap between using upper layer supervision for decomposed model and not using the supervision. We attribute this to the fact that question tokens are fewer than that of the passage, thus having less distance variations.

5.2 Inference Cost

The decomposed model enables caching of text representations that can be computed offline. While a full-scale analysis of the detailed trade-offs in storage versus latency is beyond the scope of this



(a) Passage distance comparison

(b) Question distance comparison

Figure 5: Representation distance of BERT vs decomp-BERT and distance of BERT vs decomp-BERT w/o auxiliary loss/supervision

paper, we present a set of basic calculations to illustrate that the storage cost of caching can be substantially smaller compared to the inference cost.

Assuming a use case of evaluating one million question-passage pairs daily, we first compute the storage requirements of the representations of these passages. With the BERT-base representations we estimate this to be 226KB per passage and 226GB in total for 1 million passages. The cost of storing this data and the added compute costs and reading these passages at the current vendor rates amounts to a total of $61.7 dollars per month. To estimate inference cost, we use the compute times we obtain from our calculations and use current vendor rates for GPU workloads which amounts to $148.5 dollars to support the 1 million question-passage pair workload. The substantial reduction in cost is because the storage cost is many orders of magnitude cheaper than using GPUs. Details of these calculations are listed in the Appendix.

5.3 Other Pre-trained Transformer Models

More recently, two pre-trained Transformers XLNet (Yang et al., 2019) and RoBERTa (Liu et al., 2019) have outperformed BERT models on many NLP tasks. RoBERTa has the same architecture as BERT, it is reasonable to expect the decomposition idea to work accordingly. Although XLNet brings the segment-level recurrent into the Transformer self-attention layers that making it different from BERT, the decomposition technique is likely to be applicable due to being agnostic to the layer internals. We leave showing the effectiveness of decomposition of these two models to near future work.

6 Conclusion

Transformers have improved the effectiveness of NLP tools by their ability to incorporate large contexts effectively in multiple layers. This however imposes a significant complexity cost. In this work, we showed that modeling such large contexts may not always be necessary and leverage this insight to build a decomposition of the Transformer model that provides substantial improvements in inference speed, memory reduction, while retaining most of the original model's accuracy. This decomposition model provides a simple yet strong starting point for efficient models as NLP moves towards increasingly larger models handling wider contexts.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learn-

ing on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Patrick Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. Groupreduce: Block-wise low-rank approximation for neural language model shrinking. In *Advances in Neural Information Processing Systems*, pp. 10988–10998, 2018.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019a.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does bert look at? an analysis of berts attention. *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 2019b. doi: 10.18653/v1/w19-4828. URL http://dx.doi.org/10.18653/v1/w19-4828.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N19-1423.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pp. 1379–1387, 2016.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Yaru Hao, Li Dong, Furu Wei, and Ke Xu. Visualizing and understanding the effectiveness of bert, 2019.

Shankar Iyer, Nikhil Dandekar, and Kornl Csernai. Quora question pairs. https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs, 2019.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

NVIDIA Performance Lab. Considerations for scaling gpu-ready data centers. https://www.nvidia.com/content/g/pdfs/GPU-Ready-Data-Center-Tech-Overview.pdf, 2019.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

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

Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Dawei Song, and Ming Zhou. A tensorized transformer for language modeling. *arXiv preprint arXiv:1906.09777*, 2019.

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

Jonas Močkus. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pp. 400–404. Springer, 1975.

Nogueira, Fernando. Bayesianoptimization. https://github.com/fmfn/BayesianOptimization, 2019. [Online; accessed 22-September-2019].

Google Cloud Platform. Cloud pricing. https://cloud.google.com/compute/all-pricing#gpus, https://cloud.google.com/storage/pricing, 2019.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Victor Sanh, Lysandre Debut, and Thomas Wolf. Introducing distilbert, a distilled version of bert. https://medium.com/huggingface/distilbert-8cf3380435b5, 2019.

Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

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

Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. doi: 10.18653/v1/p19-1452. URL http://dx.doi.org/10.18653/v1/p19-1452.

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

Adina Williams, Nikita Nangia, and Samuel Bowman. 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, Volume 1 (Long Papers)*, pp. 1112–1122. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/N18-1101.

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

Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and pattern Recognition*, pp. 1984–1992, 2015.

A APPENDIX

Data centers often use GPUs for inference workloads (Lab, 2019), we use the GPUs by default for both models. We use g_u to denote the cost of using one GPU per hour, n_{seq} to stand for the number of input sequences to process, b for the GPU batch size, and t_b is the time (in seconds) take to process b sequences, s denotes the storage size of the cached representations, s_u denotes the cost of storage per month, r_u is the cost of performing 10,000 reading operations (such as loading cached representations from the disk).

The total cost of the original model $Cost_{original}$ is the cost of using GPUs and is given by the formula as below:

$$Cost_{original} = t_b \cdot \frac{n_{seq}}{b} \cdot \frac{g_u}{3600}$$

And the total cost of the decomposed model $Cost_{decomp}$ includes three parts: using GPUs, storing representations on disk and loading them into memory. It is formulated as:

$$Cost_{decomp} = t_b \cdot \frac{n_{seq}}{b} \cdot \frac{g_u}{3600} + \frac{n_{seq}}{b} \cdot \frac{r_u}{10,000} + \frac{s \cdot s_u}{30 * 24 * 3600}$$

We assume a passage has 150 tokens on average (The number is calculated based on the SQuAD dataset).

We take one cloud service provider (Platform, 2019) to instantiate g_u, s_u, and r_u: one Tesla V100 GPU (16GB memory) costs \$2.48 USD per hour ($g_u = 2.48$), 1GB storage takes \$0.02 per month ($s_u = 0.02$) and additional \$0.004 per 10,000 read operations ($r_u = 0.004$)[6].

[6]Class B operations on GCP

It takes 226KB to store the vectors for 150 tokens [7], and the total storage for 1 million sequences is 226GB. The Tesla V100 GPU allows a maximum batch size of 640[8]. We measure the $t_b = 4.6$ for the original BERT-base model and $t_b = 1.4$ for the decomposed BERT-base model. Then $Cost_{original} = 30 * 4.6 * 1,000,000/640 * 2.48/3600 = \$148.5$, and $Cost_{decomp} = 30 * 1.4 * 1,000,000/640 * 2.48/3600 + 30 * 1,000,000/10,000 * 0.004 + 226 * 0.02 = \$61.7$.

[7] vector dimension=768, bfloat16 format

[8] >640 batch size will cause V100 GPU out of memory