BERT-of-Theseus: Compressing BERT by Progressive Module Replacing

Canwen Xu *
Wuhan University
Hubei, China
xucanwen@whu.edu.cn

Wangchunshu Zhou *
Beihang University
Beijing, China
zhouwangchunshu@buaa.edu.cn

Tao Ge
Microsoft Research Asia
Beijing, China
tage@microsoft.com

Furu Wei
Microsoft Research Asia
Beijing, China
fuwei@microsoft.com

Ming Zhou
Microsoft Research Asia
Beijing, China
mingzhou@microsoft.com

ABSTRACT

In this paper, we propose a novel model compression approach to effectively compress BERT by progressive module replacing. Our approach first divides the original BERT into several modules and builds their compact substitutes. Then, we randomly replace the original modules with their substitutes to train the compact modules to mimic the behavior of the original modules. We progressively increase the probability of replacement through the training. In this way, our approach brings a deeper level of interaction between the original and compact models, and smooths the training process. Compared to the previous knowledge distillation approaches for BERT compression, our approach leverages only one loss function and one hyper-parameter, liberating human effort from hyper-parameter tuning. Our approach outperforms existing knowledge distillation approaches on GLUE benchmark, showing a new perspective of model compression.[2]

1 Introduction

With the prevalence of deep learning, many huge neural models have been proposed and achieve state-of-the-art performance in various fields [12, 38]. Specifically, in Natural Language Processing (NLP), pretraining and fine-tuning have become the new norm of most tasks. Transformer-based pretrained models [4, 21, 42, 31, 6] have dominated the field of both Natural Language Understanding (NLU) and Natural Language Generation (NLG). These models benefit from their "overparameterized" nature [24] and contain millions or even billions of parameters, making it computationally expensive and inefficient considering both memory consumption and high latency. This drawback enormously hinders the applications of these models in production.

To resolve this problem, many techniques have been proposed to compress a neural network. Generally, these techniques can be categorized into Quantization [10], Weights Pruning [11] and Knowledge Distillation (KD) [14]. Among them, KD has received much attention for compressing pretrained language models. KD exploits a large *teacher* model to "teach" a compact *student* model to mimic the teacher's behavior. In this way, the knowledge embedded in the teacher model can be transferred into the smaller model. However, the retained performance of the student model relies on a well-designed distillation loss function which forces the student model to behave as the teacher. Recent studies on KD [33, 15] even leverage more sophisticated model-specific distillation loss functions for better performance.

Different from previous KD studies which explicitly exploit a distillation loss to minimize the distance between the teacher model and the student model, we propose a new genre of model compression. Inspired by the famous thought experiment "Ship of Theseus[3]" in Philosophy, where all components of a ship are gradually replaced by new ones until

---

[2]Code and pretrained model are available at https://github.com/JetRunner/BERT-of-Theseus
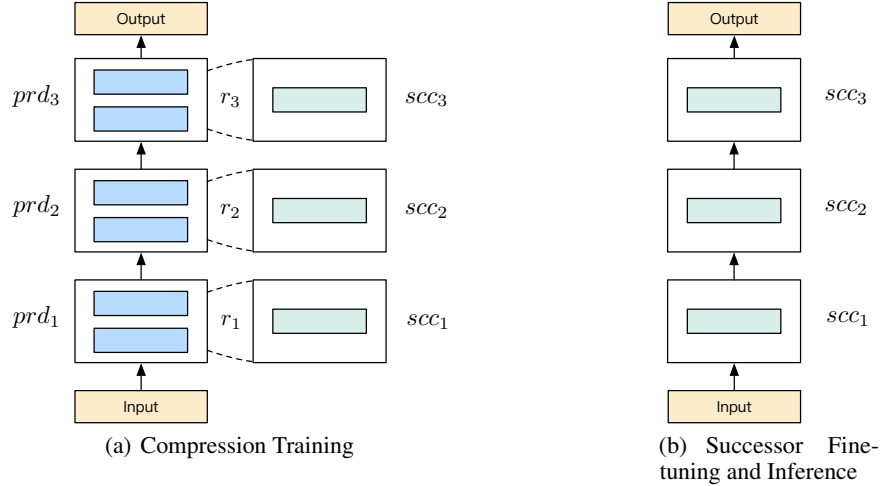
[3]https://en.wikipedia.org/wiki/Ship_of_Theseus

Figure 1: The workflow of BERT-of-Theseus. In this example, we compress a 6-layer predecessor $P = \{prd_1, .., prd_3\}$ to a 3-layer successor $S = \{scc_1, .., scc_3\}$. prd_i and scc_i contain two and one layer, respectively. (a) During module replacing training, each predecessor module prd_i is replaced with corresponding successor module scc_i by the probability of p. (b) During successor fine-tuning and inference, all successor modules $scc_{1..3}$ are combined for calculation.

no original component exists, we propose **Theseus Compression** for BERT (**BERT-of-Theseus**), which progressively substitutes modules of BERT with modules of fewer parameters. We call the original model and compressed model *predecessor* and *successor*, in correspondence to the concepts of *teacher* and *student* in KD, respectively. As shown in Figure 1, we first specify a substitute (successor module) for each predecessor module (i.e., modules in the predecessor model). Then, we randomly replace each predecessor module with its corresponding successor module by a probability and make them work together in the training phase. After convergence, we combine all successor modules to be the successor model for inference. In this way, the large predecessor model can be compressed into the compact successor model.

Theseus Compression shares a similar idea with KD, which encourages the compressed model to behave like the original, but holds many merits. First, we only use the task-specific loss function in the compression process. However, KD-based methods use task-specific loss, together with one or multiple distillation losses as its optimization objective. The use of only one loss function throughout the whole compression process allows us to unify the different phases and keep the compression in a total end-to-end fashion. Also, selecting various loss functions and balancing the weights of each loss for different tasks and datasets are always laborious [33, 28]. Second, different from recent work [15], Theseus Compression does not use Transformer-specific features for compression thus is potential to compress a wide spectrum of models. Third, instead of using the original model only for inference in KD, our approach allows the predecessor model to work in association with the compressed successor model, enabling a deeper gradient-level interaction and a smoother training process. Moreover, the different module permutations mixing both predecessor and successor modules adds extra regularization, similar to Dropout [32]. With a Curriculum Learning [1] driven replacement scheduler, our approach achieves great performance compressing BERT [4], a large pretrained Transformer model.

To summarize, our contribution is two-fold: (1) We propose a novel approach, Theseus Compression, revealing a new pathway to model compression, with only one loss function and one hyper-parameter. (2) Our compressed BERT model is $1.94\times$ faster while retaining more than 98% performance of the original model, outperforming other KD-based compression baselines.

2 Related Work

Model Compression. Model compression aims to reduce the size and computational cost of a large model while retaining as much performance as possible. Conventional explanations [3, 43] claim that the large number of weights is necessary for the training of deep neural network but a high degree of redundancy exists after training. Recent work [8] proposes The Lottery Ticket Hypothesis claiming that dense, randomly-initialized and feed-forward networks contain subnetworks that can be recognized and trained to get a comparable test accuracy to the original network.

Quantization [10] reduces the number of bits used to represent a number in a model. Weights Pruning [11, 13] conducts a binary classification to decide which weights to be trimmed from the model. Knowledge Distillation (KD) [14] aims to train a compact model which behaves like the original one. FitNets [27] demonstrates that "hints" learned by the large model can benefit the distillation process. Born-Again Neural Network [9] reveals that ensembling multiple identical-parameterized students can outperform a teacher model. LIT [17] introduces block-wise intermediate representation training. Liu et al. [20] distilled knowledge from ensemble models to improve the performance of a single model on NLU tasks. Tan et al. [35] exploited KD for multi-lingual machine translation. Different from KD-based methods, our proposed Theseus Compression is the first approach to mix the original model and compact model for training. Also, no additional loss is used throughout the whole compression procedure which eliminates the tricky hyper-parameter tuning for various losses.

Faster BERT. Very recently, many attempts have been made to speed up a large pretrained language model (e.g., BERT [4]). Michel et al. [22] reduced the parameters of a BERT model by pruning unnecessary heads in the Transformer. Shen et al. [29] quantized BERT to 2-bit using Hessian information. Also, substantial modification has been made to Transformer architecture. Fan et al. [7] exploited a structure dropping mechanism to train a BERT-like model which is resilient to pruning. ALBERT [18] leverages matrix decomposition and parameter sharing. However, these models cannot exploit ready-made model weights and require a full retraining. Tang et al. [36] used a BiLSTM architecture to extract task-specific knowledge from BERT. DistilBERT [28] applies a naive Knowledge Distillation on the same corpus used to pretrain BERT. Patient Knowledge Distillation (PKD) [33] designs multiple distillation losses between the module hidden states of the teacher and student models. Pretrained Distillation [37] pretrains the student model with a self-supervised masked LM objective on a large corpus first, then performs a standard KD on supervised tasks. TinyBERT [15] conducts the Knowledge Distillation twice with data augmentation. MobileBERT [34] devises a more computationally efficient architecture and applies knowledge distillation with a bottom-to-top layer training procedure.

3 BERT-of-Theseus

In this section, we introduce module replacing, the technique proposed for BERT-of-Theseus. Further, we introduce a Curriculum Learning driven scheduler to obtain better performance. The workflow is shown in Figure 1.

3.1 Module Replacing

The basic idea of Theseus Compression is very similar to KD. We want the successor model to act like a predecessor model. KD explicitly defines a loss to measure the similarity of the teacher and student. However, the performance greatly relies on the design of the loss function [14, 33, 15]. This loss function needs to be combined with task-specific loss [33, 17]. Different from KD, Theseus Compression only requires one task-specific loss function (e.g., Cross Entropy), which closely resembles a fine-tuning procedure. Inspired by Dropout [32], we propose module replacing, a novel technique for model compression. We call the original model and the target model *predecessor* and *successor*, respectively. First, we specify a successor module for each module in the predecessor. For example, in the context of BERT compression, we let one Transformer layer to be the successor module for two Transformer layers. Consider a predecessor model P which has n modules and a successor model S which has n predefined modules. Let $P = \{prd_1, .., prd_n\}$ denote the predecessor model, prd_i and scc_i denote the the predecessor modules and their corresponding substitutes, respectively. The output vectors of the i-th module is denoted as $\mathbf{y}_i$. Thus, the forward operation can be described in the form of:

$$\mathbf{y}_{i+1} = prd_i(\mathbf{y}_i) \tag{1}$$

During compression, we apply module replacing. First, for $(i+1)$-th module, r_{i+1} is an independent Bernoulli random variable which has probability p to be 1 and $1-p$ to be 0.

$$r_{i+1} \sim \text{Bernoulli}(p) \tag{2}$$

Then, the output of the $(i+1)$-th model is calculated as:

$$y_{i+1} = r_{i+1} * scc_i(\mathbf{y}_i) + (1 - r_{i+1}) * prd_i(\mathbf{y}_i) \tag{3}$$

where $*$ denotes the element-wise multiplication, $r_{i+1} \in \{0, 1\}$. In this way, the predecessor modules and successor modules work together in the training. Since the permutation of the hybrid model is random, it adds extra noises as a regularization for the training of the successor, similar to Dropout [32].

During training, similar to a fine-tuning process, we optimize a regular task-specific loss, e.g., Cross Entropy:

$$L = - \sum_{j \in |X|} \sum_{c \in C} [\mathbb{1} [\mathbf{z}_j = c] \cdot \log P (\mathbf{z}_j = c | \mathbf{x}_j)] \tag{4}$$
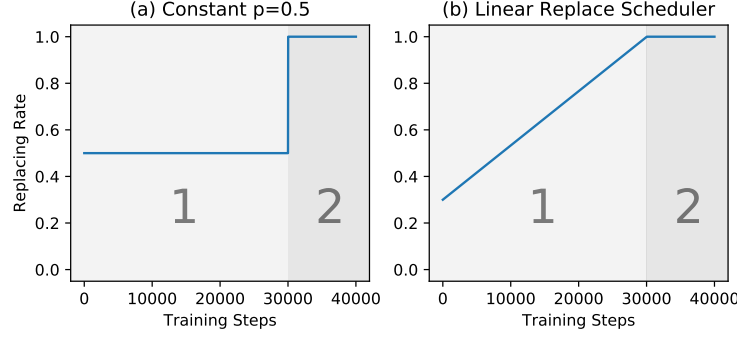
Figure 2: The replacing curves of a constant module replace rate and a replacement scheduler. We use different shades of gray to mark the two phases of Theseus Compression: (1) Module replacing. (2) Successor fine-tuning.

where $\mathbf{x}_j \in X$ is the i-th training sample; $\mathbf{z}_j$ is its corresponding ground-truth label; c and C denote a class label and the set of class labels, respectively. For back-propagation, the weights of all predecessor modules are frozen and only weights of successor will be updated. For both the embedding layer and output layer of the predecessor model are weight-frozen and directly adopted for the successor model in this training phase. In this way, the gradient can be calculated across both the predecessor and successor modules, allowing the interaction on a deeper level.

3.2 Successor Fine-tuning and Inference

Since all successor modules have not been combined for training yet, we further carry out a post-replacement fine-tuning phase. After the replacing compression converges, we collect all successor modules and combine them to be the successor model S:

$$S = \{scc_1, .., scc_n\}$$
$$\mathbf{y}_{i+1} = scc_i(\mathbf{y}_i) \tag{5}$$

Since each scc_i is smaller than prd_i in size, the predecessor model P is in essence compressed into a smaller model S. Then, we fine-tune the successor model by optimizing the same loss of Equation 4. The whole procedure including module replacing and successor fine-tuning is illustrated in Figure 2(a). Finally, we use the fine-tuned successor for inference as Equation 5.

3.3 Curriculum Replacement

Although setting a constant replacement rate p can meet the need for compressing a model, we further highlight a Curriculum Learning [1] driven replacement scheduler, which helps progressively substitute the modules in a model. Similar to Curriculum Dropout [23], we devise a replacement scheduler to dynamically tune the replacement rate p.

Here, we leverage a simple linear scheduler $\theta(t)$ to output the dynamic replacement rate p_d for step t.

$$p_d = \min(1, \theta(t)) = \min(1, kt + b) \tag{6}$$

where $k > 0$ is the coefficient and b is the basic replacement rate. The replacing rate curve with a replacement scheduler is illustrated in Figure 2(b).

In this way, we unify the two previously separated training stages and encourage an end-to-end easy-to-hard learning process. First, with more predecessor modules present, the model would more likely to correctly predict thus have a relatively small cross-entropy loss, which is helpful for smoothing the learning process. Then, at a later time of compression, more modules can be present together, encouraging the model to gradually learn to predict with less guidance from the predecessor and steadily transit to the successor fine-tuning stage.

Second, at the beginning of the compression, when $\theta(t) < 1$, considering the average learning rate for all n successor modules, the expected number of replaced modules is $n \cdot p_d$ and the expected average learning rate is:

$$lr' = (np_d/n)lr = (kt + b)lr \tag{7}$$

where lr is the constant learning rate set for the compression and lr' is the equivalent learning rate considering all successor modules. Thus, when applying a replacement scheduler, a warm-up mechanism [25] is essentially adopted at the same time, which helps the training of a Transformer.

4

Table 1: Comparison of different BERT compression approaches. "CE" and "MSE" stand for Cross Entropy and Mean Square Error, respectively. "KD" indicates the loss is for Knowledge Distillation. "CE_{TASK}", "CE_{MLM}" and 'CE_{NSP}" indicate Cross Entropy calculated on downstream tasks, Masked LM pretraining and Next Sentence Prediction, respectively. Other loss functions are described in their corresponding papers.

Method	# Layer	# Param.	Loss Function	External Data Used?	Model-Agnostic?
BERT-base [4]	12	110M	$CE_{MLM} + CE_{NSP}$	-	-
Fine-tuning	6	66M	CE_{TASK}	✗	✓
Vanilla KD [14]	6	66M	$CE_{KD} + CE_{TASK}$	✗	✓
BERT-PKD [33]	6	66M	$CE_{KD} + PT_{KD} + CE_{TASK}$	✗	✓
DistillBERT [28]	6	66M	$CE_{KD} + Cos_{KD} + CE_{MLM}$	✓ (unlabeled)	✓
PD-BERT [37]	6	66M	$CE_{MLM} + CE_{KD} + CE_{TASK}$	✓ (unlabeled)	✓
TinyBERT [15]	4	15M	$MSE_{attn} + MSE_{hidn} + MSE_{embd} + CE_{KD}$	✓ (unlabeled + labeled)	✗ (Transformer only)
MobileBERT [34]	24	25M	$FMT+AT+PKT+CE_{KD}+CE_{MLM}$	✓ (unlabeled)	✗ (BERT only)
BERT-of-Theseus (Ours)	6	66M	CE_{TASK}	✗	✓

4 Experiments

In this section, we introduce the experiments of Theseus Compression for BERT [4] compression. We compare BERT-of-Theseus with other compression methods and further conduct experiments to analyze the results.

4.1 Datasets

We evaluate our proposed approach on GLUE benchmark [39]. Specifically, we test on Microsoft Research Paraphrase Matching (MRPC) [5], Quora Question Pairs (QQP)[4] and STS-B [2] for Paraphrase Similarity Matching; Stanford Sentiment Treebank (SST-2) [30] for Sentiment Classification; Multi-Genre Natural Language Inference Matched (MNLI-m), Multi-Genre Natural Language Inference Mismatched (MNLI-mm) [41], Question Natural Language Inference (QNLI) [26] and Recognizing Textual Entailment (RTE) [39] for the Natural Language Inference (NLI) task; The Corpus of Linguistic Acceptability (CoLA) [40] for Linguistic Acceptability. Note that we exclude WNLI [19] from GLUE since the original BERT [4] paper excludes this task as well for convergence problems.

The accuracy is used as the metric for SST-2, MNLI-m, MNLI-mm, QNLI and RTE. The F1 and accuracy are used for MRPC and QQP. The Pearson correlation and Spearman correlation are used for STS-B. Matthew's correlation is used for CoLA. The results reported for the test set of GLUE are in the same format as on the official leaderboard. For the sake of comparison with [28], on dev set of GLUE, the result of MNLI is the average result on MNLI-m and MNLI-mm; the results on MRPC and QQP are reported with the average of F1 and accuracy; the result reported on STS-B is the average of the Pearson correlation and Spearman correlation.

4.2 Experimental Settings

We test our approach under a task-specific compression setting [33, 37] instead of a pretraining compression setting [28, 34]. That is to say, we use no external unlabeled corpus but only the training set of each task in GLEU to compress the model. The reason why we test our model under such a setting is that we intend to straightforwardly verify the effectiveness of our generic compression approach. The fast training process of task-specific compression (e.g., no longer than 20 GPU hours for any task of GLUE) computationally enables us to conduct more analytical experiments. For comparision, DistillBERT [28] takes 720 GPU hours to train. Plus, in real-world applications, this setting provides with more flexibility when selecting from different pretrained LMs (e.g., BERT, RoBERTa [21]) for various downstream tasks and it is easy to adopt a newly released model, without a time-consuming pretraining compression.

On the other hand, we acknowledge that a general-purpose compressed BERT can better facilitate the downstream applications in the community since it requires less computational resource to simply fine-tune a small model than compressing a large one. Thus, we release a general-purpose compressed BERT as well.

Formally, we define the task of compression as trying to retain as much performance as possible when compressing the officially released BERT-base (uncased)[5] to a 6-layer compact model with the same hidden size, following the settings

---

[4]https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs
[5]https://github.com/google-research/bert

Table 2: Experimental results on the dev set of GLUE. The numbers under each dataset indicate the number of training samples.

Method	CoLA (8.5K)	MNLI (393K)	MRPC (3.7K)	QNLI (105K)	QQP (364K)	RTE (2.5K)	SST-2 (67K)	STS-B (5.7K)	Macro Score
Teacher / Predecessor Model									
BERT-base [4]	54.3	83.5	89.5	91.2	89.8	71.1	91.5	88.9	82.5
Models Compressed with External Data									
DistillBERT [28]	43.6	79.0	87.5	85.3	84.9	59.9	90.7	81.2	76.5
PD-BERT [37]	-	**83.0**	87.2	89.0	89.1	66.7	91.1	-	-
Models Compressed without External Data									
Fine-tuning	43.4	80.1	86.0	86.9	87.8	62.1	89.6	81.9	77.2
Vanilla KD [14]	45.1	80.1	86.2	88.0	88.1	64.9	90.5	84.9	78.5
BERT-PKD [33]	45.5	81.3	85.7	88.4	88.4	66.5	91.3	86.2	79.2
BERT-of-Theseus	**51.1**	82.3	**89.0**	**89.5**	**89.6**	**68.2**	**91.5**	**88.7**	**81.2**

Table 3: Experimental results on the test set from GLUE server. The numbers under each dataset indicate the number of training samples.

Method	CoLA (8.5K)	MNLI-m/mm (393K)	MRPC (3.7K)	QNLI (105K)	QQP (364K)	RTE (2.5K)	SST-2 (67K)	STS-B (5.7K)	Macro Score
Teacher / Predecessor Model									
BERT-base [4]	52.1	84.6 / 83.4	88.9 / 84.8	90.5	71.2 / 89.2	66.4	93.5	87.1 / 85.8	80.0
Model Compressed with External Data									
PD-BERT [37]	-	**82.8 / 82.2**	86.8 / 81.7	88.9	70.4 / 88.9	65.3	91.8	-	-
Models Compressed without External Data									
Fine-tuning	41.5	80.4 / 79.7	85.9 / 80.2	86.7	69.2 / 88.2	63.6	90.7	82.1 / 80.0	75.6
Vanilla KD [14]	42.9	80.2 / 79.8	86.2 / 80.6	88.3	70.1 / 88.8	64.7	91.5	82.1 / 80.3	76.4
BERT-PKD [33]	43.5	81.5 / 81.0	85.0 / 79.9	89.0	70.7 / 88.9	65.5	92.0	83.4 / 81.6	77.0
BERT-of-Theseus	**47.8**	82.4 / 82.1	**87.6 / 83.2**	**89.6**	**71.6 / 89.3**	**66.2**	**92.2**	**85.6 / 84.1**	**78.6**

in [28, 33, 37]. Under this setting, the compressed model has 24M parameters for the token embedding (identical to the original model) and 42M parameters for the Transformer layers and obtains a $1.94\times$ speed-up for inference.

4.3 Training Details

We fine-tune BERT-base as the predecessor model for each task with the batch size of 32, the learning rate of 2×10^{-5}, and the number of epochs as 4. As a result, we are able to obtain a predecessor model with comparable performance with that reported in previous studies [28, 33, 15].

Afterward, for training successor models, following [28, 33], we use the first 6 layers of BERT-base to initialize the successor model since the over-parameterized nature of Transformer [38] could cause the model unable to converge while training on small datasets. During module replacing, We fix the batch size as 32 for all evaluated tasks to reduce the search space. All r variables only sample once for a training batch. The maximum sequence length is set to 256 on QNLI and 128 for the other tasks. We perform grid search over the sets of learning rate lr as {1e-5, 2e-5}, the basic replacing rate b as {0.1, 0.3}, the scheduler coefficient k making the dynamic replacing rate increase to 1 within the first {1000, 5000, 10000, 30000} training steps. We apply an early stopping mechanism and select the model with the best performance on the dev set. We conduct our experiments on a single Nvidia V100 16GB GPU. The peak memory usage is identical to fine-tuning a BERT-base, since there would be at most 12 layers training at the same time. The training time for each task varies depending on the different sizes of training sets. For example, it takes 20 hours to train on MNLI but less than 30 minutes on MRPC.

Table 4: Experimental results of our general-purpose model on GLUE-dev.

Method	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B
BERT-base [4]	83.5	89.5	91.2	89.8	71.1	91.5	88.9
DistillBERT [28]	79.0	87.5	85.3	84.9	59.9	90.7	81.2
BERT-of-Theseus $_{MNLI}$	**82.1**	87.5	**88.8**	**88.8**	**70.1**	**91.8**	**87.8**

4.4 Baselines

As shown in Table 1, we compare the layer numbers, parameter numbers, loss function, external data usage and model agnosticism of our proposed approach to existing methods. We set up a baseline of vanilla Knowledge Distillation [14] as in [33]. Additionally, we directly fine-tune an initialized 6-layer BERT model on GLUE tasks to obtain a natural fine-tuning baseline. Under the setting of compressing 12-layer BERT-base to a 6-layer compact model, we choose BERT-PKD [33], PD-BERT [37], and DistillBERT [28] as strong baselines. Note that DistillBERT [28] is not directly comparable here since it uses a pretraining compression setting. Both PD-BERT and DistillBERT use external unlabeled corpus. We do not include TinyBERT [15] since it has a different size setting, conducts distillation twice, and leverages extra augmented data for GLUE tasks. We also exclude MobileBERT [34], due to its redesigned Transformer block and different model size. Besides, in these two studies, the loss functions are not architecture-agnostic thus limit their applications on other models.

4.5 Experimental Results

We report the experimental results on the dev set of GLUE in Table 2 and submit our predictions to the GLUE test server and obtain the results from the official leaderboard as shown in Table 3. Note that DistillBERT does not report on test set. The BERT-base performance reported on GLUE dev set is the predecessor fine-tuned by us. The results of BERT-PKD on dev set are reproduced by us using the official implementation. In the original paper of BERT-PKD, the results of CoLA and STS-B on test set are not reported, thus we reproduce these two results. Fine-tuning and Vanilla KD baselines are both implemented by us. All other results are from the original papers. The macro scores here are calculated in the same way as the official leaderboard but are not directly comparable with GLUE leaderboard since we exclude WNLI from the calculation.

Overall, our BERT-of-Theseus retains 98.4% and 98.3% of the BERT-base performance on GLUE dev set and test set, respectively. On every task of GLUE, our model dramatically outperforms the fine-tuning baseline, indicating that with the same loss function, our proposed approach can effectively transfer knowledge from the predecessor to the successor. Also, our model obviously outperforms the vanilla KD [14] and Patient Knowledge Distillation (PKD) [33], showing its supremacy over the KD-based compression approaches. On MNLI, our model performs better than BERT-PKD but slightly lower than PD-BERT [37]. However, PD-BERT exploits an additional corpus which provides much more samples for knowledge transferring. Also, we would like to highlight that on RTE, our model achieves nearly identical performance to BERT-base and on QQP our model even outperforms BERT-base. To analyze, a moderate model size may help generalize and prevent overfitting on downstream tasks. Notably, on both large datasets with more than 350K samples (e.g., MNLI and QQP) and small datasets with fewer than 4K samples (e.g., MRPC and RTE), our model can consistently achieve good performance, verifying the robustness of our approach.

4.6 General-purpose Model

Although our approach achieves good performance under a task-specific setting, it requires more memory to fine-tune a full-size predecessor than a compact BERT (e.g., DistillBERT [28]). Liu et al. [21] found that a model fine-tuned on MNLI can successfully transfer to other sentence classification tasks. Thus, we release our compressed model by conducting compression on MNLI as a general-purpose compact BERT to facilitate downstream applications[6]. After compression, we fine-tune the successor model on other sentence classification tasks and compare the results with DistillBERT [28] in Table 4. Our general-purpose model achieves an identical performance on MRPC and remarkably outperforms DistillBERT on the other sentence-level tasks.

5 Analysis

In this section, we conduct extensive experiments to analyze our BERT-of-Theseus.

---

[6]`https://huggingface.co/canwenxu/BERT-of-Theseus-MNLI`

Table 5: Impact of the replacement for different modules on GLUE-dev. $prd_i \rightarrow scc_i$ indicates the replacement of the i-th module from the predecessor.

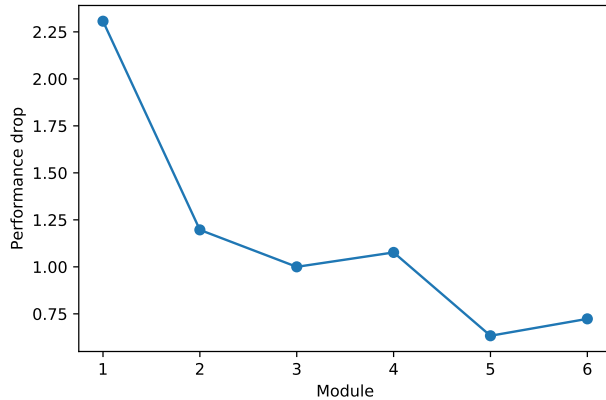| Replacement	QNLI(Δ)	MNLI(Δ)	QQP(Δ)
Predecessor	91.87	84.54	89.48
$prd_1 \rightarrow scc_1$	88.50 (-3.37)	81.89 (-2.65)	88.58 (-0.90)
$prd_2 \rightarrow scc_2$	90.54 (-1.33)	83.33 (-1.21)	88.43 (-1.05)
$prd_3 \rightarrow scc_3$	90.76 (-1.11)	83.27 (-1.27)	88.86 (-0.62)
$prd_4 \rightarrow scc_4$	90.46 (-1.41)	83.34 (-1.20)	88.86 (-0.62)
$prd_5 \rightarrow scc_5$	90.74 (-1.13)	84.16 (-0.38)	89.09 (-0.39)
$prd_6 \rightarrow scc_6$	90.57 (-1.30)	84.09 (-0.45)	89.06 (-0.42)

Figure 3: Average performance drop when replacing the predecessor module prd_i with successor module scc_i on QNLI, MNLI and QQP (dev set).

5.1 Impact of Module Replacement

As pointed out in previous work [7], different layers of a Transformer play imbalanced roles for inference. To explore the effect of different module replacements, we iteratively use one compressed successor module (constant replacing rate, without successor fine-tuning) to replace its corresponding predecessor module on QNLI, MNLI and QQP, as shown in Table 5. We illustrate the average performance drop on three tasks in Figure 3. Surprisingly, different from a similar study for the importance of different Transformer layers in [7], which is basically a U-curve, our results show that the replacement of the last two modules have only a trivial influence on the overall performance while the replacement of the first module significantly harms the performance. To analyze, the linguistic features are mainly extracted by the first few layers. Therefore, the reduced representation capability becomes the bottleneck for the following layers. On the contrary, high-quality low-level features can help the following layers, thus the reduced module size has only a limited influence on the final results.

5.2 Impact of Replacing Rate

We attempt to adopt different replacing rates on GLUE tasks. First, we fix the batch size to be 32 and learning rate lr to be 2×10^{-5} and conduct compression on each task. On the other hand, as we analyzed in Section 3.3, the equivalent learning rate lr' is affected by the replacing rate. To further eliminate the influence of learning rate, we fix the equivalent learning rate lr' to be 2×10^{-5} and adjust learning rate lr for different replacing rates by $lr = lr'/p$.

We illustrate the results with different replacing rates on two representative tasks (MRPC and RTE) in Figure 4. The trivial gap between two curves in both figures indicate that the effect of different replacing rates on equivalent learning rate is not the main factor for the performance differences. Generally speaking, BERT-of-Theseus is not very sensitive to different replacing rates. A replacing rate in the range between 0.5 and 0.7 can always lead to a satisfying performance on all GLUE tasks. However, a significant performance drop can be observed on all tasks if the replacing rate is too small (e.g., $p = 0.1$). On the other hand, the best replacing rate differs across tasks.
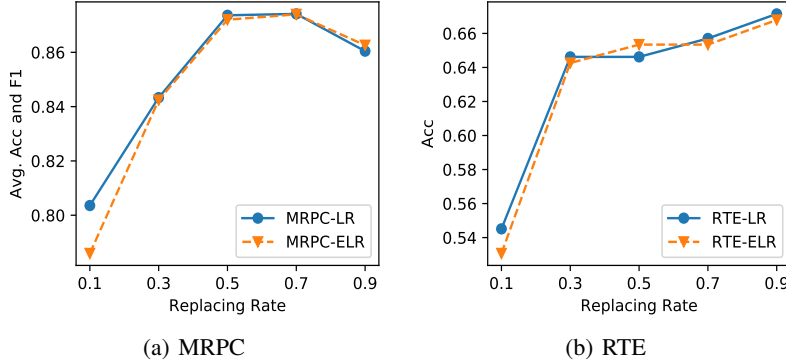
Figure 4: Performance of different replacing rate on MRPC and RTE. "LR" and "ELR" denote that the learning rate and equivalent learning rate are fixed, respectively.

Table 6: Comparison of models compressed with a constant replacing rate, a curriculum replacement scheduler and its corresponding anti-curriculum scheduler on GLUE-dev.

Strategy	CoLA(Δ)	MNLI(Δ)	MRPC(Δ)	QNLI(Δ)	QQP(Δ)	RTE(Δ)	SST-2(Δ)	STS-B(Δ)
Constant Replacing Rate	44.4	81.9	87.1	88.5	88.6	66.4	90.6	88.4
Anti-curriculum	42.8 (-1.6)	79.8 (-2.1)	85.6 (-1.5)	87.8 (-0.7)	87.6 (-1.0)	62.4 (-4.0)	88.8 (-1.8)	85.4 (-3.0)
Curriculum Scheduler	**51.1 (+6.7)**	**82.3 (+0.4)**	**89.0 (+1.9)**	**89.5 (+1.0)**	**89.6 (+1.0)**	**68.2 (+1.8)**	**91.5 (+0.9)**	**88.7 (+0.3)**

5.3 Impact of Replacement Scheduler

To study the impact of our curriculum replacement strategy, we compare the results of BERT-of-Theseus compressed with a constant replacing rate and with a replacement scheduler. The constant replacing rate for the baseline is searched over $\{0.5, 0.7, 0.9\}$. Additionally, we implement an "anti-curriculum" baseline, similar to the one in [23]. For each task, we adopt the same coefficient k and basic replacing rate b to calculate the p_d as Equation 6 for both curriculum replacement and anti-curriculum. However, we use $1 - p_d$ as the dynamic replacing rate for anti-curriculum baseline. Thus, we can determine whether the improvement of curriculum replacement is simply due to an inconstant replacing rate or an easy-to-hard curriculum design.

As shown in Table 6, our model compressed with curriculum scheduler consistently outperforms a model compressed with a constant replacing rate. On the contrary, a substantial performance drop is observed on the model compressed with an anti-curriculum scheduler, which further verifies the effectiveness of the curriculum replacement strategy.

6 Discussion and Conclusion

In this paper, we propose Theseus Compression, a novel model compression approach. We use this approach to compress BERT to a compact model, which outperforms other models compressed by Knowledge Distillation, with only one hyper-parameter, one loss function and no external data. Our work highlights a new genre of model compression and reveals a new path towards model compression.

A known limitation of our approach is that to allow a successor module to replace a predecessor module, they must have the same input and output sizes. First, given this restriction, we can still perform depth reduction (i.e., reducing the number of layers). As analyzed in [28], the hidden size dimension has a smaller impact on computational efficiency than the depth, for a fixed parameter budget. Second, there have been many developed in-place substitutes (e.g., ShuffleNet unit [44] for ResBlock [12], Reformer Layer [16] for Transformer Layer [38]), which can be directly adopted as the successor. Third, it is possible to use a feed-forward neural network to map features between the hidden spaces of different sizes [15].

Although our model has achieved good performance compressing BERT, it would be interesting to explore its possible applications in other neural models. As summarized in Table 1, our model does not rely on any model-specific features to compress BERT. Therefore, it is potential to apply Theseus Compression to other large models (e.g., ResNet [12]

in Computer Vision). For the future work, we would like to conduct Theseus Compression on Convolutional Neural Network and Graph Neural Network.

References

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, 2009.

[2] Alexis Conneau and Douwe Kiela. Senteval: An evaluation toolkit for universal sentence representations. In *LREC*. European Language Resources Association (ELRA), 2018.

[3] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *NIPS*, 2013.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[5] William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *IWP@IJCNLP*, 2005.

[6] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. Unified language model pre-training for natural language understanding and generation. In *NeurIPS*, 2019.

[7] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *ICLR*, 2020.

[8] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

[9] Tommaso Furlanello, Zachary Chase Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born-again neural networks. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1602–1611. PMLR, 2018.

[10] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization, 2014.

[11] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[13] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[15] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding, 2019.

[16] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *ICLR*, 2020.

[17] Animesh Koratana, Daniel Kang, Peter Bailis, and Matei Zaharia. LIT: Learned intermediate representation training for model compression. In *ICML*, 2019.

[18] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *ICLR*, 2020.

[19] Hector J. Levesque. The winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*. AAAI, 2011.

[20] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Improving multi-task deep neural networks via knowledge distillation for natural language understanding, 2019.

[21] 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, 2019.

[22] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In *NeurIPS*, 2019.

[23] Pietro Morerio, Jacopo Cavazza, Riccardo Volpi, René Vidal, and Vittorio Murino. Curriculum dropout. In *ICCV*, pages 3564–3572. IEEE Computer Society, 2017.

[24] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt, 2019.

[25] Martin Popel and Ondej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):4370, Apr 2018. ISSN 1804-0462. doi: 10.2478/pralin-2018-0002.

[26] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP*, pages 2383–2392. The Association for Computational Linguistics, 2016.

[27] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2015.

[28] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2019.

[29] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Q-bert: Hessian based ultra low precision quantization of bert, 2019.

[30] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

[31] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. MASS: masked sequence to sequence pre-training for language generation. In *ICML*, 2019.

[32] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.

[33] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for BERT model compression. In *EMNLP/IJCNLP*, pages 4322–4331. Association for Computational Linguistics, 2019.

[34] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: Task-agnostic compression of bert by progressive knowledge transfer, 2019.

[35] Xu Tan, Yi Ren, Di He, Tao Qin, Zhou Zhao, and Tie-Yan Liu. Multilingual neural machine translation with knowledge distillation. In *ICLR*, 2019.

[36] Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from bert into simple neural networks, 2019.

[37] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models, 2019.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

[39] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*, 2019.

[40] Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural network acceptability judgments. *TACL*, 7: 625–641, 2019.

[41] Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*, 2018.

[42] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, pages 5754–5764, 2019.

[43] Shuangfei Zhai, Yu Cheng, Zhongfei (Mark) Zhang, and Weining Lu. Doubly convolutional neural networks. In *NIPS*, 2016.

[44] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018.