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

Canwen Xu[1]*, **Wangchunshu Zhou**[2]*, **Tao Ge**[3], **Furu Wei**[3], **Ming Zhou**[3]
[1] University of California, San Diego [2] Beihang University [3] Microsoft Research Asia
[1] cxu@ucsd.edu [2] zhouwangchunshu@buaa.edu.cn
[3] {tage,fuwei,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. Compared to the previous knowledge distillation approaches for BERT compression, our approach does not introduce any additional loss function. Our approach outperforms existing knowledge distillation approaches on GLUE benchmark, showing a new perspective of model compression.[1]

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 (He et al., 2016; Vaswani et al., 2017). Specifically, in Natural Language Processing (NLP), pretraining and fine-tuning have become the new norm of most tasks. Transformer-based pretrained models (Devlin et al., 2019; Liu et al., 2019b; Yang et al., 2019; Song et al., 2019; Dong et al., 2019) have dominated the field of both Natural Language Understanding (NLU) and Natural Language Generation (NLG). These models benefit from their "overparameterized" nature (Nakkiran et al., 2020) 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 (Gong et al., 2014), Weights Pruning (Han et al., 2016) and Knowledge Distillation (KD) (Hinton et al., 2015). 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 (Sun et al., 2019; Jiao et al., 2019) 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"[2] in Philosophy, where all components of a ship are gradually replaced by new ones until 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

* Equal contribution. Work done during these two authors' internship at Microsoft Research Asia.

[1] The code and pretrained model are available at https://github.com/JetRunner/BERT-of-Theseus

[2] https://en.wikipedia.org/wiki/Ship_of_Theseus

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pages 7859–7869,
November 16–20, 2020.

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 a compact successor model.

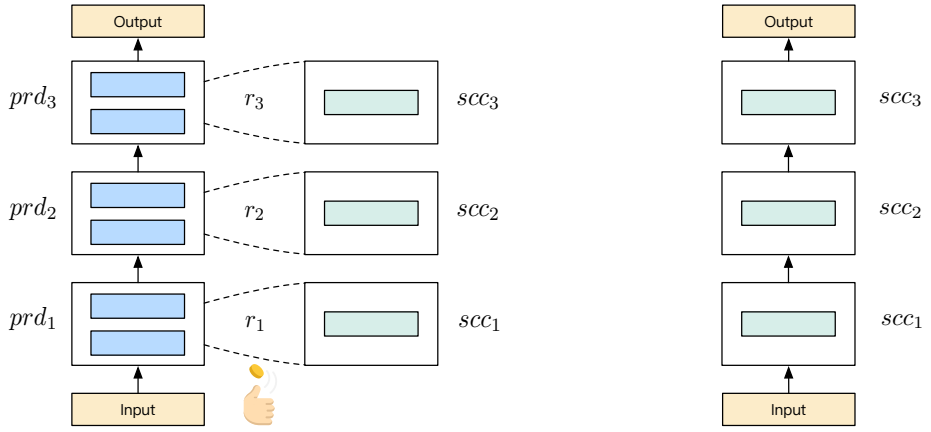
Theseus Compression shares a similar idea with KD, which encourages the compressed model to behave like the original, but holds many merits. First, we only use the task-specific loss function in the compression process. However, KD-based methods use task-specific loss, together with one or multiple distillation losses as its optimization objective. Also, selecting various loss functions and balancing the weights of each loss for different tasks and datasets can be laborious (Sun et al., 2019; Sanh et al., 2019). Second, different from recent work (Jiao et al., 2019), 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 possible gradient-level interaction. Moreover, the different module permutations mixing both predecessor and successor modules may add extra regularization, similar to Dropout (Srivastava et al., 2014). With a Curriculum Learning (Bengio et al., 2009) driven replacement scheduler, our approach achieves promising performance compressing BERT (Devlin et al., 2019), 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 no additional loss function. (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 (Denil et al., 2013; Zhai et al., 2016) 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 (Frankle and Carbin, 2019) 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 (Gong et al., 2014) reduces the number of bits used to represent a number in a model. Weights Pruning (Han et al., 2016; He et al., 2017) conducts a binary classification to decide which weights to be trimmed from the model. Knowledge Distillation (KD) (Hinton et al., 2015) aims to train a compact model which behaves like the original one. FitNets (Romero et al., 2015) demonstrates that "hints" learned by the large model can benefit the distillation process. Born-Again Neural Network (Furlanello et al., 2018) reveals that ensembling multiple identical-parameterized students can outperform a teacher model. LIT (Koratana et al., 2019) introduces block-wise intermediate representation training. Liu et al. (2019a) distilled knowledge from ensemble models to improve the performance of a single model on NLU tasks. Tan et al. (2019) 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 simplifies the implementation.

Faster BERT Very recently, many attempts have been made to speed up a large pretrained language model (e.g., BERT (Devlin et al., 2019)). Michel et al. (2019) reduced the parameters of a BERT model by pruning unnecessary heads in the Transformer. Shen et al. (2020) quantized BERT to 2-bit using Hessian information. Also, substantial modification has been made to Transformer architecture. Fan et al. (2020) exploited a structure dropping mechanism to train a BERT-like model which is resilient to pruning. ALBERT (Lan et al., 2020) leverages matrix decomposition and parameter sharing. However, these models cannot exploit ready-made model weights and require a full retraining. Tang et al. (2019) used a BiLSTM architecture to extract task-specific knowledge from BERT. DistilBERT (Sanh et al., 2019) applies a naive Knowledge Distillation on the same corpus used to pretrain BERT. Patient Knowledge Distillation (PKD) (Sun et al., 2019) designs multiple distillation losses between the module hidden states of the teacher and student models. Pretrained Distilla-



(a) Compression Training

(b) Successor Fine-tuning and Inference

Figure 1: The workflow of BERT-of-Theseus. In this example, we compress a 6-layer predecessor $P = \{prd_1, \ldots, prd_3\}$ to a 3-layer successor $S = \{scc_1, \ldots, scc_3\}$. 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 $scc_{1\ldots3}$ are combined for calculation.

tion (Turc et al., 2019) 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 (Jiao et al., 2019) conducts the Knowledge Distillation twice with data augmentation. MobileBERT (Sun et al., 2020) devises a more computationally efficient architecture and applies knowledge distillation with a bottom-to-top layer training procedure. PABEE (Zhou et al., 2020b) exploits early exiting to dynamically accelerate the inference of BERT.

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 vastly relies on the design of the loss function (Hinton et al., 2015; Sun et al., 2019; Jiao et al., 2019). This loss function needs to be combined with task-specific loss (Sun et al., 2019; Koratana et al., 2019). 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 (Srivastava et al., 2014), 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 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 $P = \{prd_1, \ldots, prd_n\}$ 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 (Srivastava et al., 2014).

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

$$L = -\sum_{j \in |X|} \sum_{c \in C} \left[\mathbb{1}\left[\mathbf{z}_j = c\right] \cdot \log P\left(\mathbf{z}_j = c | \mathbf{x}_j\right)\right] \tag{4}$$

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. 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 deeper interaction.

3.2 Successor Fine-tuning and Inference

To make the training and inference processes as close as possible, we further carry out a post-replacement fine-tuning phase to allow all successor modules to work together. After the replacing compression converges, we collect all successor modules and combine them to be the successor model S:

$$\begin{aligned} S &= \{scc_1, \ldots, scc_n\} \\ \mathbf{y}_{i+1} &= scc_i(\mathbf{y}_i) \end{aligned} \tag{5}$$

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 (Bengio et al., 2009) driven replacement scheduler, which coordinates the progressive replacement of the modules.

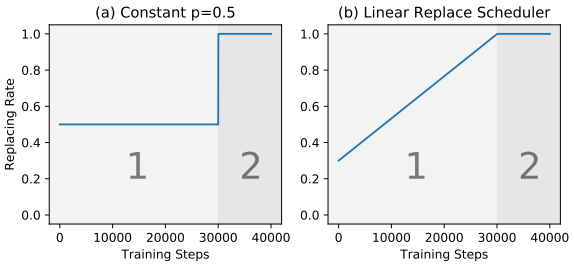


Figure 2: The replacing curves of a constant module replace rate and a replacement scheduler. We use different shades of gray to mark the two phases of Theseus Compression: (1) Module replacing. (2) Successor fine-tuning.

Similar to (Morerio et al., 2017; Zhou et al., 2020a), 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 (Popel and Bojar, 2018) is essentially adopted at the same time, which helps the training of a Transformer.

4 Experiments

In this section, we introduce the experiments of Theseus Compression for BERT (Devlin et al., 2019) 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 the GLUE benchmark (Wang et al., 2019; Dolan and Brockett, 2005; Conneau and Kiela, 2018; Socher et al., 2013; Williams et al., 2018; Rajpurkar et al., 2016; Warstadt et al., 2019). Note that we exclude WNLI (Levesque, 2011) following the original BERT paper (Devlin et al., 2019).

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 (Sanh et al., 2019), on the development set of GLUE, the result of MNLI is an average 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 and Spearman correlation.

4.2 Experimental Settings

We test our approach under a task-specific compression setting (Sun et al., 2019; Turc et al., 2019) instead of a pretraining compression setting (Sanh et al., 2019; Sun et al., 2020). That is to say, we use no external unlabeled corpus but only the training set of each task in GLUE to compress the model. The reason behind this decision 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 comparison, DistilBERT (Sanh et al., 2019) 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 (Liu et al., 2019b)) for various downstream tasks and it is easy to adopt a newly released model, without a time-consuming pretraining compression. We will also discuss the possibility to use an MNLI model for a general purpose with intermediate transfer learning (Pruksachatkun et al., 2020).

Formally, we define the task of compression as trying to retain as much performance as possible when compressing the officially released BERT-base (uncased)[3] to a 6-layer compact model with the same hidden size, following the settings in (Sanh et al., 2019; Sun et al., 2019; Turc et al., 2019). 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 (Sanh et al., 2019; Sun et al., 2019; Jiao et al., 2019).

Afterward, for training successor models, following (Sanh et al., 2019; Sun et al., 2019), we use the first 6 layers of BERT-base to initialize the successor model since the over-parameterized nature of Transformer (Vaswani et al., 2017) 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 development set. We conduct our experiments on a single Nvidia V100 16GB GPU. The peak memory usage is approximately 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

[3] https://github.com/google-research/bert

Method	# Layer	# Param.	Loss Function	External Data Used?	Model-Agnostic?
BERT-base (2019)	12	110M	CE_{MLM} + CE_{NSP}	-	-
Fine-tuning	6	66M	CE_{TASK}	✗	✓
Vanilla KD (2015)	6	66M	CE_{KD} + CE_{TASK}	✗	✓
BERT-PKD (2019)	6	66M	CE_{KD} + PT_{KD} + CE_{TASK}	✗	✓
DistilBERT (2019)	6	66M	CE_{KD} + Cos_{KD} + CE_{MLM}	✓ (unlabeled)	✓
PD-BERT (2019)	6	66M	CE_{MLM} + CE_{KD} + CE_{TASK}	✓ (unlabeled)	✓
TinyBERT (2019)	4	15M	MSE_{attn} + MSE_{hidn} + MSE_{embd} + CE_{KD}	✓ (unlabeled + labeled)	✗
MobileBERT (2020)	24	25M	FMT+AT+PKT+CE_{KD}+CE_{MLM}	✓ (unlabeled)	✗
BERT-of-Theseus (Ours)	6	66M	CE_{TASK}	✗	✓

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.

train on MNLI but less than 30 minutes on MRPC.

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 (Hinton et al., 2015) as in (Sun et al., 2019). Additionally, we directly fine-tune a truncated 6-layer BERT model (the bottom 6 layers of the original BERT)[4] 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 (Sun et al., 2019), PD-BERT (Turc et al., 2019), and DistilBERT (Sanh et al., 2019) as strong baselines. Note that DistilBERT (Sanh et al., 2019) is not directly comparable here since it uses a pretraining compression setting. Both PD-BERT and DistilBERT use external unlabeled corpus. Additionally, we use LayerDrop (Fan et al., 2020) on BERT weights to prune the model on downstream tasks. We do not include TinyBERT (Jiao et al., 2019) since it conducts distillation twice and leverages extra augmented data for GLUE tasks. We also exclude MobileBERT (Sun et al., 2020), 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 types of models.

4.5 Experimental Results

We report the experimental results on the development 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 DistilBERT does not report on the test set. The BERT-base performance reported on GLUE development set is the predecessor fine-tuned by us. The results of BERT-PKD on the development set are reproduced by us using the official implementation. In the original paper of BERT-PKD, the results of CoLA and STS-B on the 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.[5] 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 development 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 (Hinton et al., 2015) and Patient Knowledge Distillation (PKD) (Sun et al., 2019), showing its supremacy over the KD-based compression approaches. On MNLI, our model performs better than BERT-PKD but slightly lower than PD-BERT (Turc et al., 2019). However, PD-BERT exploits an additional corpus which provides much more samples for knowledge transferring. Also, we would like to highlight that

[4] We also tried the top 6 layers and interleaving 6 layers but both perform worse than the bottom 6 layers.

[5] Please note that the reported results of DistilBERT are different across various versions on arXiv. The results here are from v3, which was the newest version when we composed this paper.

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
BERT-base (2019)	54.3	83.5	89.5	91.2	89.8	71.1	91.5	88.9	82.5
DistilBERT (2019)	43.6	79.0	87.5	85.3	84.9	59.9	90.7	81.2	76.5
PD-BERT (2019)	-	**83.0**	87.2	89.0	89.1	66.7	91.1	-	-
Fine-tuning	43.4	80.1	86.0	86.9	87.8	62.1	89.6	81.9	77.2
Vanilla KD (2015)	45.1	80.1	86.2	88.0	88.1	64.9	90.5	84.9	78.5
BERT-PKD (2019)	45.5	81.3	85.7	88.4	88.4	66.5	91.3	86.2	79.2
LayerDrop (2020)	45.4	80.7	85.9	88.4	88.3	65.2	90.7	85.7	78.8
BERT-of-Theseus	**51.1**	82.3	**89.0**	**89.5**	**89.6**	**68.2**	**91.5**	**88.7**	**81.2**

Table 2: Experimental results (median of 5 runs) on the development set of GLUE. The numbers under each dataset indicate the number of training samples. All models listed above (except BERT-base) have 66M parameters, 6 layers and 1.94× speed-up.

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
BERT-base (2019)	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
PD-BERT (2019)	-	**82.8 / 82.2**	86.8 / 81.7	88.9	70.4 / 88.9	65.3	91.8	-	-
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 (2015)	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 (2019)	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**

Table 3: Experimental results on the test set from the GLUE server. All models listed above (except BERT-base) have 66M parameters, 6 layers and 1.94× speed-up.

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 Intermediate-Task Transfer Learning

Although our approach achieves good performance under a task-specific setting, it requires more computational resources to fine-tune a full-size predecessor than a compact BERT (e.g., DistilBERT (Sanh et al., 2019)). Pruksachatkun et al. (2020) found that models trained on some datasets can be used for a second-round fine-tuning. Thus, we use MNLI as the intermediate task and release our compressed model by conducting compression on MNLI to facilitate downstream applications. After compression, we fine-tune the successor model on other sentence classification tasks and compare the results with DistilBERT (Sanh et al., 2019) in Table 4. Our model achieves an identical performance on MRPC and outperforms DistilBERT on the other sentence-level tasks. Also, our intermediate-task transfer results also outperform PD-BERT (Turc et al., 2019) on three tasks, indicating that our task-specific model is also competitive for a general purpose through the intermediate-task transfer learning approach.

5 Analysis

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

5.1 Impact of Module Replacement

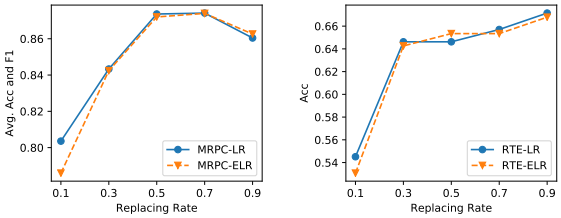
As pointed out in previous work (Fan et al., 2020), different layers of a Transformer play imbalanced roles for inference. To explore the effect of different module replacements, we iteratively use one compressed successor module (constant replacing rate, without successor fine-tuning) to replace its corresponding predecessor module on QNLI, MNLI and QQP, as shown in Table 5. Our results show that the replacement of the last two modules have limited 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

Method	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B
BERT-base (2019)	83.5	89.5	91.2	89.8	71.1	91.5	88.9
DistilBERT (2019)	79.0	**87.5**	85.3	84.9	59.9	90.7	81.2
PD-BERT (2019)	**83.0**	87.2	**89.0**	**89.1**	66.7	91.1	-
BERT-of-Theseus $_{\text{MNLI}}$	82.1	**87.5**	88.8	88.8	**70.1**	**91.8**	**87.8**

Table 4: Experimental results of intermediate-task transfer learning on GLUE-dev.

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)

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.



(a) MRPC (b) RTE

Figure 3: Performance of different replacing rate on MRPC and RTE. "LR" and "ELR" denote that the learning rate and equivalent learning rate are fixed, respectively.

few layers. Therefore, the reduced representation capability becomes the bottleneck for the following layers.

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 the learning rate, we fix the equivalent learning rate lr' to be 2×10^{-5} and adjust the 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 3. 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. 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.

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 (Morerio et al., 2017). 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. In contrast, a substantial performance drop is observed on the model compressed with an anti-curriculum scheduler, which further verifies the effectiveness and importance of the curriculum replacement strategy.

5.4 Impact of Predecessor Layers

We further replace different numbers of Transformer layers with one layer to verify the effectiveness of Theseus Compression under different settings. We replace 3/4 layers with one Transformer layer, resulting in a 4/3-layer BERT model.

Strategy	CoLA(Δ)	MNLI(Δ)	MRPC(Δ)	QNLI(Δ)	QQP(Δ)	RTE(Δ)	SST-2(Δ)	STS-B(Δ)
Constant 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	**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)**

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

Method	#Layer	Speed-up	CoLA (8.5K)	MNLI (393K)	MRPC (3.7K)	QNLI (105K)	QQP (364K)	RTE (2.5K)	SST-2 (67K)	STS-B (5.7K)	Macro Score
BERT-base (2019)	12	1.00×	54.3	83.5	89.5	91.2	89.8	71.1	91.5	88.9	82.5
Fine-tuning	6	1.94×	43.4	80.1	86.0	86.9	87.8	62.1	89.6	81.9	77.2
BERT-of-Theseus	6	1.94×	**51.1**	**82.3**	**89.0**	**89.5**	**89.6**	**68.2**	**91.5**	**88.7**	**81.2**
Fine-tuning	4	2.82×	33.9	78.4	86.0	82.3	87.1	58.2	87.2	78.4	73.9
BERT-of-Theseus	4	2.82×	**41.3**	**80.0**	**87.5**	**86.1**	**88.7**	**61.9**	**89.1**	**82.5**	**77.2**
Fine-tuning	3	3.66×	27.5	78.1	81.9	80.4	86.5	57.7	85.9	76.8	71.9
BERT-of-Theseus	3	3.66×	**35.0**	**78.8**	**84.3**	**82.1**	**87.3**	**59.5**	**87.2**	**78.9**	**74.1**

Table 7: Experimental results of replacing different numbers of layers with one layer on GLUE-dev. "#Layer" indicates the number of layers in the compressed models.

The results are shown in Table 7. BERT-of-Theseus consistently outperforms the fine-tuned truncated BERT baselines, demonstrating its effectiveness under different settings.

6 Discussion

In this paper, we propose Theseus Compression, a novel model compression approach. We use this approach to compress BERT to a compact model that outperforms other models compressed by Knowledge Distillation. Our work highlights a new genre of model compression and reveals a new path towards model compression.

For future work, we would like to explore the possibility of applying Theseus Compression on heterogeneous network modules. First, many developed in-place substitutes (e.g., ShuffleNet unit (Zhang et al., 2018) for ResBlock (He et al., 2016), Reformer Layer (Kitaev et al., 2020) for Transformer Layer (Vaswani et al., 2017)) are natural successor modules that can be directly adopted in Theseus Compression. Also, it is possible to use a feed-forward neural network to map features between the hidden spaces of different sizes (Jiao et al., 2019) to enable replacement between modules with different input and output sizes. 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 (He et al., 2016) in Computer Vision). In addition, we would like to conduct Theseus Compression on more types of neural networks including Convolutional Neural Networks and Graph Neural Networks. We will also investigate the combination of our compression-based approach with recently proposed dynamic acceleration method (Zhou et al., 2020b) to further improve the efficiency of pre-trained language models.

Acknowledgments

We are grateful for the insightful comments from the anonymous reviewers. Tao Ge is the corresponding author.

References

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

Alexis Conneau and Douwe Kiela. 2018. Senteval: An evaluation toolkit for universal sentence representations. In *LREC*.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of

deep bidirectional transformers for language understanding. In *NAACL-HLT*.

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

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

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

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

Tommaso Furlanello, Zachary Chase Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. 2018. Born-again neural networks. In *ICML*.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*.

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

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

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

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

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

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

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

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

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

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019a. Improving multi-task deep neural networks via knowledge distillation for natural language understanding. *arXiv preprint arXiv:1904.09482*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. 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? In *NeurIPS*.

Pietro Morerio, Jacopo Cavazza, Riccardo Volpi, René Vidal, and Vittorio Murino. 2017. Curriculum dropout. In *ICCV*.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. 2020. Deep double descent: Where bigger models and more data hurt. In *ICLR*.

Martin Popel and Ondřej Bojar. 2018. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70.

Yada Pruksachatkun, Jason Phang, Haokun Liu, Phu Mon Htut, Xiaoyi Zhang, Richard Yuanzhe Pang, Clara Vania, Katharina Kann, and Samuel R. Bowman. 2020. Intermediate-task transfer learning with pretrained language models: When and why does it work? In *ACL*.

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

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

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-BERT: hessian based ultra low precision quantization of BERT. In *AAAI*.

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

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

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

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for BERT model compression. In *EMNLP-IJCNLP*.

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

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

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*.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*.

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

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

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

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

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

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

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

Wangchunshu Zhou, Tao Ge, Ke Xu, Furu Wei, and Ming Zhou. 2020a. Scheduled drophead: A regularization method for transformer models. *arXiv preprint arXiv:2004.13342*.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020b. Bert loses patience: Fast and robust inference with early exit. In *NeurIPS*.