A Flexible Multi-Task Model for BERT Serving

Anonymous ACL submission

Abstract

We present an efficient BERT-based multi-task (MT) framework that is particularly suitable for iterative and incremental development of the tasks. The proposed framework is based on the idea of partial fine-tuning, i.e. only fine-tune some top layers of BERT while keep the other layers frozen. For each task, we train independently a single-task (ST) model using partial fine-tuning. Then we compress the task-specific layers in each ST model using knowledge distillation. Those compressed ST models are finally merged into one MT model so that the frozen layers of the former are shared across the tasks. We exemplify our approach on eight GLUE tasks, demonstrating that it is able to achieve 99.6% of the performance of the full fine-tuning method, while reducing up to two thirds of its overhead.

1 Introduction

In this work we explore the strategies of BERT (Devlin et al., 2019) serving for multiple tasks under the following two constraints: 1) Memory and computational resources are limited. On edge devices such as mobile phones, this is usually a hard constraint. On local GPU stations and Cloud-based servers, this constraint is not as hard but it is still desirable to reduce the computation overhead to cut the serving cost. 2) The tasks are expected to be modular and are subject to frequent updates. When one task is updated, the system should to be able to quickly adapt to the task modification such that the other tasks are not affected. This is a typical situation for applications (e.g. AI assistant) under iterative and incremental development.

In principle, there are two strategies of BERT serving: *single-task serving* and *multi-task serving*. In single-task serving, one independent single-task model is trained and deployed for each task. Typically, those models are obtained by fine-tuning a copy of the pre-trained BERT and are completely different from each other. Single-task serving has the advantage of being flexible and modular as there is no dependency between the task models. The downside is its inefficiency in terms of both memory usage and computation, as neither parameters nor computation are shared or reused across the tasks. In multi-task serving, one single multi-task model is trained and deployed for all tasks. This model is typically trained with multi-task learning (MTL) (Caruana, 1997; Ruder, 2017). Compared to its single-task counterpart, multi-task serving is much more computationally efficient and incurs much less memory usage thanks to its sharing mechanism. However, it has the disadvantage in that any modification made to one task usually affect the other tasks.

The main contribution of this work is the proposition of a framework for BERT serving that simultaneously achieves the flexibility of single-task serving and the efficiency of multi-task serving. Our method is based on the idea of partial fine-tuning, i.e. only fine-tuning some topmost layers of BERT depending on the task and keeping the remaining bottom layers frozen. The fine-tuned layers are task-specific, which can be updated on a per-task basis. The frozen layers at the bottom, which plays the role of a feature extractor, can be shared across the tasks.

2 Related Work

The standard practice of using BERT is *fine-tuning*, i.e. the entirety of the model parameters is adjusted on the training corpus of the downstream task, so that the model is adapted to that specific task (Devlin et al., 2019). There is also an alternative *feature-based* approach, used by ELMo (Peters et al., 2018). In the latter approach, the pre-trained model is regarded as a feature extractor with *frozen parameters*. During the learning of a downstream task, one feeds a fixed or learnable combination of the model's intermediate representations as input to

L	QNLI	RTE	QQP	MNLI	SST-2	MRPC	CoLA	STS-B
1	85.9	60.3	86.1	77.1	91.6	77.2	38.7	84.8
2	88.3	63.5	88.3	80.8	91.9	80.6	40.0	86.1
3	89.9	65.3	89.0	82.5	91.2	84.6	45.3	87.3
4	90.7	69.0	89.7	83.3	92.0	84.3	48.6	88.2
5	91.0	**71.5**	90.1	84.0	92.2	**89.7**	51.3	88.3
6	91.2	71.1	90.3	84.2	93.1	86.8	53.1	86.4
7	91.3	70.0	90.5	83.9	93.0	87.5	51.5	88.6
8	91.5	70.8	90.6	84.5	92.8	88.0	**55.2**	88.9
9	91.6	70.8	90.7	84.0	92.5	87.7	54.7	88.8
10	**91.7**	69.7	**91.1**	84.5	93.0	87.3	55.0	88.7
11	**91.7**	70.4	**91.1**	84.5	93.1	88.2	54.7	**89.1**
12	91.6	69.7	**91.1**	**84.6**	**93.4**	88.2	54.7	88.8

Table 1: Dev results on GLUE datasets obtained with partial fine-tuning. The parameter L indicates the number of fine-tuned transformer layers. For each dataset and for each value of L, we always run the experiment 5 times with different initializations and report the maximum dev result obtained. The best result in each column is highlighted in bold face. Shaded numbers indicate that they attain 99% of the best result of the column. It can be seen that although fine-tuning more layers generally leads to better performance, the benefit of doing so suffers diminishing returns. Perhaps surprisingly, for RTE, MRPC and CoLA it is the partial fine-tuning with roughly half of the layers frozen that gives the best results.

the task-specific module, and only the parameters of the latter will be updated. It has been shown that the fine-tuning approach is generally superior to the feature-based approach for BERT in terms of task performance (Devlin et al., 2019; Peters et al., 2019).

A natural middle ground between these two approaches is *partial fine-tuning*, i.e. only fine-tuning some topmost layers of BERT while keeping the remaining bottom layers frozen. This approach has been studied in (Houlsby et al., 2019; Merchant et al., 2020), where the authors observed that fine-tuning only the top layers can almost achieve the performance of full fine-tuning on several GLUE tasks. The approach of partial fine-tuning essentially regards the bottom layers of BERT as a feature extractor. Freezing weights from bottom layers is a sensible idea as previous studies show that the mid layer representations produced by BERT are most transferrable, whereas the top layers representations are more task-oriented (Wang et al., 2019; Tenney et al., 2019b,a; Liu et al., 2019a; Merchant et al., 2020).

3 Method

In what follows, we denote by $\mathcal{T}$ the set of all target tasks. We always use the 12-layer version of BERT as the pre-trained language model. The proposed framework features a pipeline (Fig. 1) that consists of three steps: 1) Single task partial fine-tuning; 2) Single task knowledge distillation; 3) Model merging. We give details of these steps below.

3.1 Single Task Partial Fine-Tuning

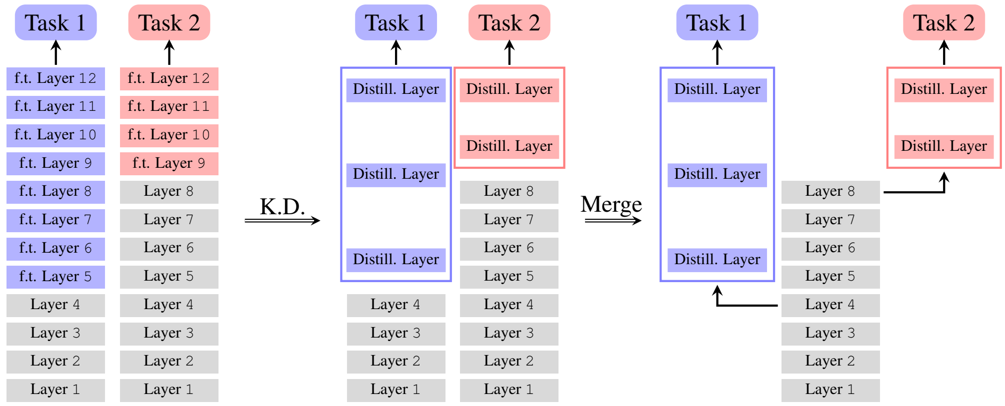
In the first step, we partial fine-tune for each task an independent copy of BERT. The exact number of layers L to fine-tune is a hyper-parameter and may vary across the tasks. We propose to experiment for each task with different value of L within range $N_{\min} \leqslant L \leqslant N_{\max}$, and select the one that gives the best validation performance. The purpose of imposing the search range $[N_{\min}, N_{\max}]$ is to guarantee a minimum degree of parameter sharing. In the subsequent experiments on GLUE tasks (see Section 4.3), we set $N_{\min} = 4$ and $N_{\max} = 10$.

This step produces a collection of single-task models as depicted in Fig. 1(a). We shall refer to them single-task *teacher models*, as they are to be knowledge distilled to further reduce the memory and computation overhead.

3.2 Single Task Knowledge Distillation

As there is no interaction between the tasks, the process of knowledge distillation (KD) can be carried out separately for each task. In principle any of the existing KD methods for BERT (Wang et al., 2020; Aguilar et al., 2020; Sun et al., 2019a; Jiao et al., 2020; Xu et al., 2020a) suits our needs. In preliminary experiments we found out that as long as the student model is properly initialized, the vanilla knowledge distillation (Hinton et al., 2015) can be as performant as those more sophisticated methods.

Assume that the teacher model for task $\tau \in \mathcal{T}$ contains $L^{(\tau)}$ fine-tuned layers at the top and $12 - L^{(\tau)}$ frozen layers at the bottom. Our goal is to compress the former into a smaller $l^{(\tau)}$-layer module. The proposed initialization scheme is very simple: we initialize the student model with the weights from the corresponding layers of the teacher. More precisely, let N_s denote the number of layers (including both frozen and task-specific layers) in the student, where $N_s < 12$. We propose to initialize the student from the bottommost N_s layers of the teacher. The value of $l^{(\tau)}$, i.e. the number of task-specific layers in the student model for task τ, determines the final memory and computation overhead for that task.



(a) Teacher models (b) Student models (c) Final multi-task model

Figure 1: Pipeline of the proposed method. (a) For each task we train separately a task-specific model with partial fine-tuning, i.e. only the weights from some topmost layers (blue and red blocks) of the pre-trained model are updated while the rest are kept frozen (gray blocks). (b) We perform knowledge distillation independently for each task on the task-specific layers of the teacher models. (c) The student models are merged into one MT model so that the frozen layers of the former can be shared.

3.3 Model Merging

In the final step, we merge the single-task student models into one multi-task model (Fig. 1(c)) so that the parameters and computations carried out in the frozen layers can be shared. To achieve this, it suffices to load weights from multiple model checkpoints into one computation graph.

4 Experiments

In this section, we compare the performance and efficiency of our model with various baselines on eight GLUE tasks.

4.1 Metrics

The performance metrics for GLUE tasks is accuracy except for CoLA and STS-B. We use Matthews correlation for CoLA, and Pearson correlation for STS-B.

To measure the parameter and computational efficiency, we introduce the *total number of transformer layers* that are needed to perform inference for all eight tasks. For the models studied in our experiments, the actual memory usage and the computational overhead are approximately linear with respect to this number. It is named "overhead" in the header of Table 2.

4.2 Baselines

The baseline models/methods can be divided into 4 categories:

Single-task without KD. There is only one method in this category, i.e. the standard practice of single task *full fine-tuning* that creates a separate model for each task.

Single-task with KD. The methods in this category create a separate model for each task, but a certain knowledge distillation method is applied to compress each task model into a 6-layer one. The KD methods include (Hinton et al., 2015; Xu et al., 2020b; Sanh et al., 2019; Turc et al., 2019; Sun et al., 2019b; Jiao et al., 2020; Wang et al., 2020).

Multi-task learning. This category includes two versions of MT-DNN (Liu et al., 2019b, 2020), both of which produce one single multi-task model. 1) *MT-DNN (full)* is jointly trained for all eight tasks. It corresponds to the idea scenario where all tasks are known in advance. 2) *MT-DNN (LOO)*, where "LOO" stands for "leave-one-out", corresponds to the scenario where one of the eight tasks is *not* known in advance. The model is jointly pre-trained on the 7 available tasks. Then an output layer for the "unknown" task is trained with the pre-trained weights frozen.

Flexible multi-task. Our models under various efficiency constraints. *Ours (w/o KD)* means that no knowledge distillation is applied to the task models. The number of fine-tuned layers for each task is selected according to the criterion described in Section 3.1. *Ours (KD-n)* means that knowledge distillation is applied such that the student model for each task contains exactly n task-specific layers. For *Ours (mixed)*, we determine the number of task-specific layers for each task based on the marginal benefit (in terms of task performance met-

	QNLI	RTE	QQP	MNLI	SST-2	MRPC	CoLA	STS-B	Avg.	Layers	Overhead
Full fine-tuning	91.6	69.7	91.1	**84.6**	93.4	88.2	54.7	88.8	82.8	12×8	96 (100%)
DistillBERT[b]	89.2	59.9	88.5	82.2	91.3	87.5	51.3	86.9	79.6	6×8	48 (50.0%)
Vanilla-KD[c]	88.0	64.9	88.1	80.1	90.5	86.2	45.1	84.9	78.5	6×8	48 (50.0%)
PD-BERT[d]	89.0	66.7	89.1	83.0	91.1	87.2	-	-	-	6×8	48 (50.0%)
BERT-PKD[e]	88.4	66.5	88.4	81.3	91.3	85.7	45.5	86.2	79.2	6×8	48 (50.0%)
BERT-of-Theseus[f]	89.5	68.2	89.6	82.3	91.5	89.0	51.1	88.7	81.2	6×8	48 (50.0%)
TinyBERT[g]	90.5	72.2	90.6	83.5	91.6	88.4	42.8	-	-	6×8	48 (50.0%)
MiniLM[h]	88.4	66.5	88.4	81.3	91.3	85.7	45.5	86.2	79.2	6×8	48 (50.0%)
MT-DNN (full)[j]	91.1	**80.9**	87.6	84.4	**93.5**	87.4	51.3	86.8	82.9	12×1	**12** (12.5%)
MT-DNN (LOO)[k]	69.7	60.6	66.5	56.7	79.2	74.2	10.2	72.9	-	-	-
Ours (KD-1)	86.4	66.1	91.0	77.5	90.7	85.1	36.4	88.3	77.4	$7+1{\times}8$	15 (15.6%)
Ours (KD-2)	88.6	64.6	**91.3**	81.7	92.7	86.3	44.0	88.6	79.7	$7+2{\times}8$	23 (24.0%)
Ours (KD-3)	90.2	66.8	91.2	82.9	92.7	88.0	50.0	**88.9**	81.3	$7+3{\times}8$	31 (32.3%)
Ours (w/o KD)	**91.7** (2,10)	71.5 (7,5)	91.1 (2,10)	84.5 (4,8)	93.1 (6,6)	**89.7** (7,5)	**55.2** (4,8)	**88.9** (4,8)	**83.2**	$7+60$	67 (69.8%)
Ours (mixed)	90.2 (2,3)	71.5 (7,5)	91.0 (2,1)	82.9 (4,3)	92.7 (6,2)	88.0 (7,3)	**55.2** (4,8)	88.3 (4,1)	82.5	$7+26$	33 (34.3%)

Table 2: A comparison of performance and overhead between our approach and various baselines (see §4.2 for more details). The best result in each column is highlighted in bold face. Shaded numbers indicate that they attain 99% of the *Full fine-tuning* baseline. Results of $[b]$ are from (Sanh et al., 2019); $[c]$-$[f]$ are from (Xu et al., 2020b); $[g]$-$[h]$ are from (Wang et al., 2020); $[j]$-$[k]$ are reproduced by us with the toolkit from (Liu et al., 2020). Round bracket (x, y) indicates that the underlying task model before merging consists of x frozen layers and y task-specific layers (fine-tuned or knowledge-distilled). In the "Layers" column, notation $7 + 2 \times 8$ implies that in the final multi-task model there are 7 shared frozen layers and 2 task-specific layers for each of the 8 task.

ric) of adding more layers to the task. More precisely, for each task we keep adding task-specific layers as long as the marginal benefit of doing so is no less than a pre-determined threshold c. In Table 2, we report the result for $c = 1.0$. Results with other values of c can be found in appendices.

4.3 Results

The results are summarized in Table 2. From the table it can be seen that the proposed method *Ours (mixed)* outperforms all KD methods while being more efficient. Compared to the single-task full fine-tuning baseline, our method reduces up to around two thirds of the total overhead while achieves 99.6% of its performance.

We observe that MT-DNN (full) achieves the best average performance with the lowest overhead. However, its performance superiority primarily comes from one big boost on a single task (RTE) rather than consistent improvements on all tasks. In fact, we see that MT-DNN (full) suffers performance degradation on QQP and STS-B due to *task interference*, a known problem for MTL (Caruana, 1997; Bingel and Sogaard, 2017; Alonso and Plank, 2017; Wu et al., 2020). From our perspective, the biggest disadvantage of MT-DNN is that it assumes full knowledge of all target tasks in advance. From the results of MT-DNN (LOO), we observe that MT-DNN has difficulty in handling new tasks if the model is not allowed to be retrained.

4.4 Discussions

One major advantage of the proposed architecture is its flexibility. First, different tasks may be fed with representations from different layers of BERT, which encapsulate different levels of linguistic information (Liu et al., 2019a). On QQP we achieve an accuracy of 91.0, outperforming all KD baselines with *merely one* task-specific layer that is connected to the 2nd layer of BERT. Second, our architecture explicitly allows for allocating uneven resources to different tasks. We have redistributed the resources among the tasks in *ours (mixed)*, resulting in both greater performance and efficiency. Third, our framework does not compromise the modular design of the system. The model can be straightforwardly updated on on a per-task basis.

5 Conclusion

We have presented our framework that is designed to provide efficient and flexible BERT-based multi-task serving. We have demonstrated on eight GLUE datasets that the proposed method achieves both strong performance and efficiency. We will release our code and hope that it can facilitate BERT serving in cost-sensitive applications.

References

Gustavo Aguilar, Yuan Ling, Yu Zhang, Benjamin Yao, Xing Fan, and Chenlei Guo. 2020. Knowledge distillation from internal representations. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7350–7357. AAAI Press.

Héctor Alonso and Barbara Plank. 2017. When is multitask learning effective? semantic sequence prediction under varying data conditions. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 44–53, Valencia, Spain. Association for Computational Linguistics.

Joachim Bingel and Anders Sogaard. 2017. Identifying beneficial task relations for multi-task learning in deep neural networks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 164–169, Valencia, Spain. Association for Computational Linguistics.

Rich Caruana. 1997. Multitask Learning. *Machine Learning*, 28(1):41–75. 00000.

Kevin Clark, Minh-Thang Luong, Urvashi Khandelwal, Christopher D. Manning, and Quoc V. Le. 2019. BAM! born-again multi-task networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5931–5937, Florence, Italy. Association for Computational Linguistics.

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

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. 2019a. Linguistic knowledge and transferability of contextual representations. 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)*, pages 1073–1094, Minneapolis, Minnesota. Association for Computational Linguistics.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019b. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496, Florence, Italy. Association for Computational Linguistics.

Xiaodong Liu, Yu Wang, Jianshu Ji, Hao Cheng, Xueyun Zhu, Emmanuel Awa, Pengcheng He, Weizhu Chen, Hoifung Poon, Guihong Cao, and Jianfeng Gao. 2020. The Microsoft toolkit of multitask deep neural networks for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 118–126, Online. Association for Computational Linguistics.

Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, and Ian Tenney. 2020. What happens to BERT embeddings during fine-tuning? In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 33–44, Online. Association for Computational Linguistics.

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

Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 7–14, Florence, Italy. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098.

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

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019a. Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4323–4332, Hong Kong, China. Association for Computational Linguistics.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019b. Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4323–4332, Hong Kong, China. Association for Computational Linguistics.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019a. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, and Ellie Pavlick. 2019b. What do you learn from context? probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations*.

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

Alex Wang, Jan Hula, Patrick Xia, Raghavendra Pappagari, R. Thomas McCoy, Roma Patel, Najoung Kim, Ian Tenney, Yinghui Huang, Katherin Yu, Shuning Jin, Berlin Chen, Benjamin Van Durme, Edouard Grave, Ellie Pavlick, and Samuel R. Bowman. 2019. Can you tell me how to get past sesame street? sentence-level pretraining beyond language modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4465–4476, Florence, Italy. Association for Computational Linguistics.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Sen Wu, Hongyang R Zhang, and Christopher Ré. 2020. Understanding and improving information transfer in multi-task learning. *arXiv preprint arXiv:2005.00944*.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020a. BERT-of-theseus: Compressing BERT by progressive module replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7859–7869, Online. Association for Computational Linguistics.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020b. BERT-of-theseus: Compressing BERT by progressive module replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7859–7869, Online. Association for Computational Linguistics.

Supplementary Material

Hyper-parameter tuning

The approach presented in this work introduces two new hyper-parameters for each task $\tau \in \mathcal{T}$, namely the number of fine-tuned layers $L^{(\tau)}$ for the teacher and the number of knowledge distilled layer $l^{(\tau)}$ for the student. If the resources permit, these two hyper-parameters should be tuned separately for each task. As introduced in Section 3.1, we suggest to constrain L within the range $4 \leq L^{(\tau)} \leq 10$. As for $l^{(\tau)}$ which determines the eventual task-specific overhead, we impose $l^{(\tau)} \leq 3$. Since we always determines $L^{(\tau)}$ first, we do not need to experiment with every combination of $(L^{(\tau)}, l^{(\tau)})$. Combining these together, our approach requires approximately 10x (7 for L and 3 for l) more training time compared to conventional full fine-tuning approach. Although 10x more training time is admittedly significant, in practice the cost is manageable (typically 2 or 3 days per task on a single Nvidia Tesla V100 GPU).

Detailed Experiment Results

In the box plots of Figure 2 above we report the performance of the student models initialized from pre-trained BERT and from the teacher. It can be clearly seen that the latter initialization scheme generally outperforms the former. Besides, we also observe that although increasing the number of task-specific layers improves the performance, the marginal benefit of doing so varies across tasks. Notably, for QQP and STS-B the student models with only one task-specific layer are able to attain 99% of the performance of their teacher.

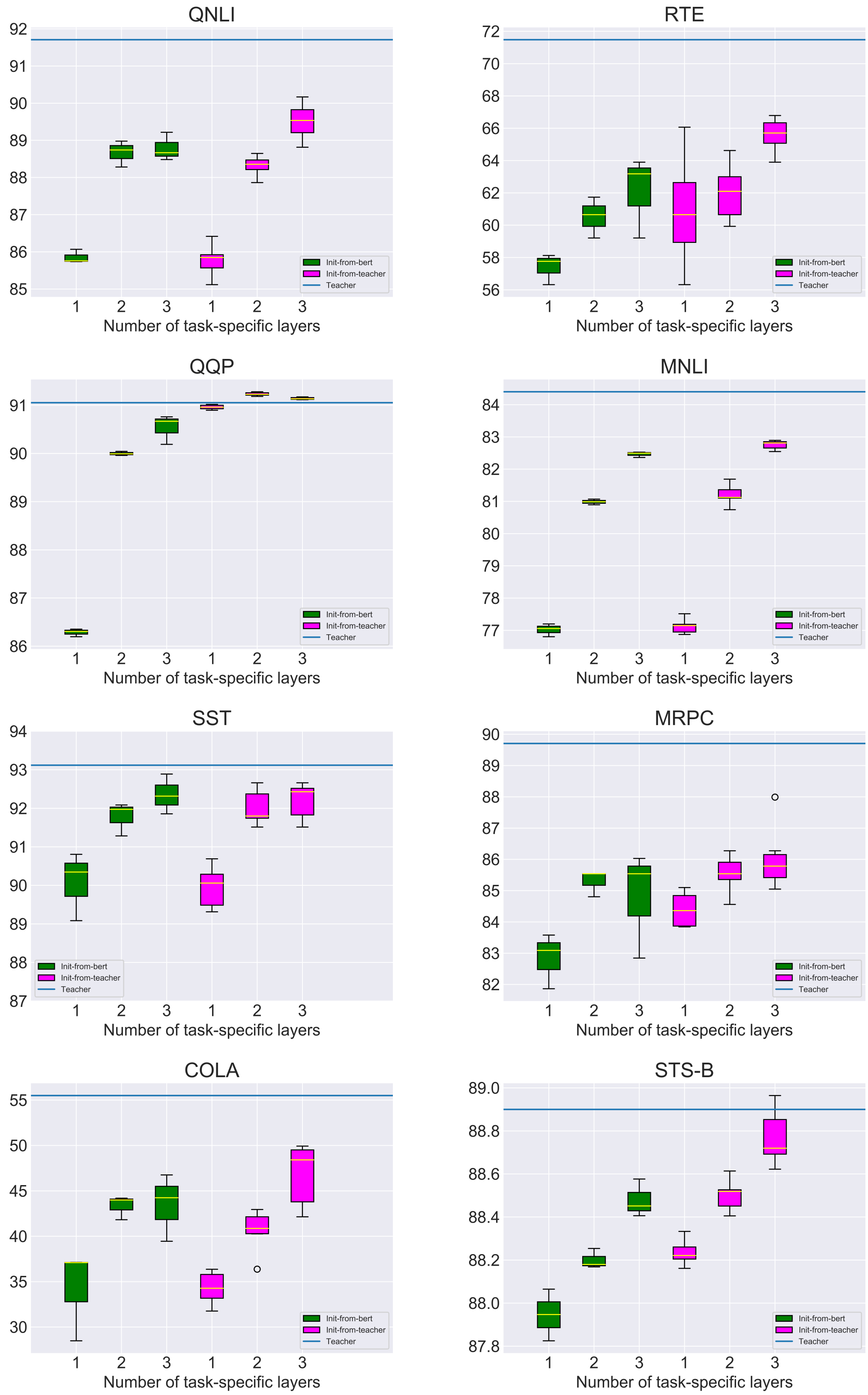


Figure 2: A comparison of the task performance between vanilla initialization (initialize from pre-trained BERT) and teacher initialization as described in Section 3.2 for $n \in \{1, 2, 3\}$, where n is the number of task-specific layers in the student model.

Performance-Efficiency Trade-off

In Fig 3, we report the performance of our method with various values of c, where c is defined as the minimal marginal benefit (in terms of task performance metric) that every task-specific layer should bring (see Section 4.2).

Industrial Application

We have implemented our framework in the application of utterance understanding of a commercial AI assistant. Our flexible multi-task model forms the bulk of the utterance understanding system, which processes over 100 million user queries per day with a peak throughput of nearly 4000 queries-per-second (QPS).

For each user query, the utterance understanding system performs various tasks, including emotion recognition, incoherence detection, domain classification, intent classification, named entity recognition, slot filling, etc. Due to the large workload, these tasks are developed and maintained by a number of different teams. As the AI assistant itself is under iterative/incremental development, its utterance understanding system undergoes frequent updates[1]:

- Update of training corpus, e.g. when new training samples become available or some mislabeled samples are corrected or removed.
- Redefinition of existing tasks. For instance, when a more fine-grained intent classification is needed, we may need to redefine existing intent labels or introduce new labels.
- Introduction of new tasks. This may happen when the AI assistant needs to upgrade its skillsets so as to perform new tasks (e.g. recognize new set of instructions, play verbal games with kids, etc).
- Removal of obsolete tasks. Sometimes a task is superseded by another task, or simply deprecated due to commercial considerations. Those tasks need to be removed from the system.

One imperative feature for the system is the *modular design*, i.e. the tasks should be independent of each other so that any modification made to one task should does not affect the other tasks. Clearly, a conventional multi-task system does not meet our need as multi-task training breaks modularity.

Before the introduction of BERT, our utterance understanding system is based on single-task serving, i.e. a separate model is deployed for each task. As those models are relatively lightweight (TextCNN/LSTM), overhead is not an issue. However, with the introduction of BERT, the cost for single-task serving becomes a valid concern as each task model (a unique 12-layer fine-tuned BERT) requires two Nvidia Tesla V100 GPUs for stable serving that meets the latency requirement.

With the primary objective of reducing cost, we have implemented the proposed flexible multi-task model in our utterance understanding system, which provides serving for a total of 21 downstream tasks. Overall, there are 40 transformer layers of which 8 are shared frozen layers (on average 1.5 task-specific layers per task). Using only 5 Nvidia Tesla V100 GPUs, we achieve a P99 latency of 32 ms under a peak throughput of 4000 QPS. Compared with single-task serving for 21 tasks which would require 42 GPUs, we estimate that our system reduces the total serving cost by up to 88%.

[1] Not necessarily frequent for any particular task, but overall frequent if we regard the system as a whole.

	QNLI	RTE	QQP	MNLI	SST-2	MRPC	CoLA	STS-B	Avg.	Layers	Overhead
Full fine-tuning	91.6	69.7	91.1	**84.6**	93.4	88.2	54.7	88.8	82.8	12×8	96 (100%)
Ours (KD-1)	86.4	66.1	91.0	77.5	90.7	85.1	36.4	88.3	77.4	$7 + 1{\times}8$	15 (15.6%)
Ours (KD-2)	88.6	64.6	**91.3**	81.7	92.7	86.3	44.0	88.6	79.7	$7 + 2{\times}8$	23 (24.0%)
Ours (KD-3)	90.2	66.8	91.2	82.9	92.7	88.0	50.0	**88.9**	81.3	$7 + 3{\times}8$	31 (32.3%)
Ours ($c = 1.0$)	90.2	71.5	91.0	82.9	92.7	88.0	**55.2**	88.3	82.5	$7 + 26$	33 (34.3%)
	(2,3)	(7,5)	(2,1)	(4,3)	(6,2)	(7,3)	(4,8)	(4,1)			
Ours ($c = 2.0$)	88.6	66.1	91.0	81.7	92.7	85.1	50.0	88.3	80.4	$7 + 13$	20 (20.2%)
	(2,2)	(7,1)	(2,1)	(4,2)	(6,2)	(7,1)	(4,3)	(4,1)			
Ours ($c = 3.0$)	86.4	66.1	91.0	81.7	90.7	85.1	50.0	88.3	79.9	$7 + 11$	18 (18.8%)
	(2,1)	(7,1)	(2,1)	(4,2)	(6,1)	(7,1)	(4,3)	(4,1)			
Ours (w/o KD)	**91.7**	71.5	91.1	84.5	93.1	**89.7**	**55.2**	**88.9**	**83.2**	$7 + 60$	67 (69.8%)
	(2,10)	(7,5)	(2,10)	(4,8)	(6,6)	(7,5)	(4,8)	(4,8)			

Table 3: Results with various values of c. This parameter controls the performance-efficiency trade-off of the overall multi-task model, in the sense that we allow the growth of an existing task module by one more task-specific layer only if that would bring a performance gain greater than c.