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
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.

<table>
<thead>
<tr>
<th>$L$</th>
<th>QNLI</th>
<th>RTE</th>
<th>QQP</th>
<th>MNLI</th>
<th>SST-2</th>
<th>MRPC</th>
<th>CoLA</th>
<th>STS-B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>85.9</td>
<td>60.3</td>
<td>86.1</td>
<td>77.1</td>
<td>91.6</td>
<td>77.2</td>
<td>38.7</td>
<td>84.8</td>
</tr>
<tr>
<td>2</td>
<td>88.3</td>
<td>63.5</td>
<td>88.3</td>
<td>80.8</td>
<td>91.9</td>
<td>80.6</td>
<td>40.0</td>
<td>86.1</td>
</tr>
<tr>
<td>3</td>
<td>89.9</td>
<td>65.3</td>
<td>89.0</td>
<td>82.5</td>
<td>91.2</td>
<td>84.6</td>
<td>45.3</td>
<td>87.3</td>
</tr>
<tr>
<td>4</td>
<td>90.7</td>
<td>69.0</td>
<td>89.7</td>
<td>83.3</td>
<td>92.0</td>
<td>84.1</td>
<td>48.6</td>
<td>88.2</td>
</tr>
<tr>
<td>5</td>
<td>91.0</td>
<td>71.5</td>
<td>90.1</td>
<td>84.0</td>
<td>92.2</td>
<td>89.7</td>
<td>51.3</td>
<td>88.3</td>
</tr>
<tr>
<td>6</td>
<td>91.2</td>
<td>71.1</td>
<td>90.3</td>
<td>84.2</td>
<td>93.3</td>
<td>86.8</td>
<td>53.1</td>
<td>86.4</td>
</tr>
<tr>
<td>7</td>
<td>91.3</td>
<td>70.0</td>
<td>90.5</td>
<td>83.9</td>
<td>93.0</td>
<td>87.5</td>
<td>51.5</td>
<td>88.6</td>
</tr>
<tr>
<td>8</td>
<td>91.5</td>
<td>70.8</td>
<td>90.6</td>
<td>84.5</td>
<td>92.8</td>
<td>88.0</td>
<td>55.2</td>
<td>88.9</td>
</tr>
<tr>
<td>9</td>
<td>91.6</td>
<td>70.8</td>
<td>90.7</td>
<td>84.0</td>
<td>92.5</td>
<td>87.7</td>
<td>54.7</td>
<td>88.8</td>
</tr>
<tr>
<td>10</td>
<td>91.7</td>
<td>69.7</td>
<td>91.1</td>
<td>84.5</td>
<td>93.0</td>
<td>87.3</td>
<td>55.0</td>
<td>88.7</td>
</tr>
<tr>
<td>11</td>
<td>91.7</td>
<td>70.4</td>
<td>91.1</td>
<td>84.5</td>
<td>93.1</td>
<td>88.2</td>
<td>54.7</td>
<td>89.1</td>
</tr>
<tr>
<td>12</td>
<td>91.6</td>
<td>69.7</td>
<td>91.1</td>
<td>84.6</td>
<td>93.4</td>
<td>88.2</td>
<td>54.7</td>
<td>88.8</td>
</tr>
</tbody>
</table>

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 $T$ 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 values of $L$ within range $N_{\text{min}} \leq L \leq N_{\text{max}}$, and select the one that gives the best validation performance. The purpose of imposing the search range $[N_{\text{min}}, N_{\text{max}}]$ is to guarantee a minimum degree of parameter sharing. In the subsequent experiments on GLUE tasks (see Section 4.3), we set $N_{\text{min}} = 4$ and $N_{\text{max}} = 10$.

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 $\tau \in T$ 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 $\tau$, determines the final memory and computation overhead for that task.
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).

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-
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). 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 $c$ 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 Soggaard, 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


Supplementary Material

Hyper-parameter tuning

The approach presented in this work introduces two new hyper-parameters for each task $τ \in T$, namely the number of fine-tuned layers $L^{(τ)}$ for the teacher and the number of knowledge distilled layer $l^{(τ)}$ 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 $4 \leq L^{(τ)} \leq 10$. As for $l^{(τ)}$ which determines the eventual task-specific overhead, we impose $l^{(τ)} \leq 3$. Since we always determine $L^{(τ)}$ first, we do not need to experiment with every combination of $(L^{(τ)}, l^{(τ)})$. 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:

- 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%.

---

1Not necessarily frequent for any particular task, but overall frequent if we regard the system as a whole.
<table>
<thead>
<tr>
<th></th>
<th>QNLI</th>
<th>RTE</th>
<th>QQP</th>
<th>MNLI</th>
<th>SST-2</th>
<th>MRPC</th>
<th>CoLA</th>
<th>STS-B</th>
<th>Avg.</th>
<th>Layers</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full fine-tuning</td>
<td>91.6</td>
<td>69.7</td>
<td>91.1</td>
<td>84.6</td>
<td>93.4</td>
<td>88.2</td>
<td>54.7</td>
<td>88.8</td>
<td>82.8</td>
<td>12 × 8</td>
<td>96 (100%)</td>
</tr>
<tr>
<td>Ours (KD-1)</td>
<td>86.4</td>
<td>66.1</td>
<td>91.0</td>
<td>77.5</td>
<td>90.7</td>
<td>85.1</td>
<td>36.4</td>
<td>88.3</td>
<td>77.4</td>
<td>7 + 1 × 8</td>
<td>15 (15.6%)</td>
</tr>
<tr>
<td>Ours (KD-2)</td>
<td>88.6</td>
<td>64.6</td>
<td>91.3</td>
<td>81.7</td>
<td>92.7</td>
<td>86.3</td>
<td>44.0</td>
<td>88.6</td>
<td>79.7</td>
<td>7 + 2 × 8</td>
<td>23 (24.0%)</td>
</tr>
<tr>
<td>Ours (KD-3)</td>
<td>90.2</td>
<td>66.8</td>
<td>91.2</td>
<td>82.9</td>
<td>92.7</td>
<td>88.0</td>
<td>50.0</td>
<td>88.9</td>
<td>81.3</td>
<td>7 + 3 × 8</td>
<td>31 (32.3%)</td>
</tr>
<tr>
<td>Ours (c = 1.0)</td>
<td>90.2</td>
<td>71.5</td>
<td>91.0</td>
<td>82.9</td>
<td>92.7</td>
<td>88.0</td>
<td>55.2</td>
<td>88.3</td>
<td>82.5</td>
<td>7 + 26</td>
<td>33 (34.3%)</td>
</tr>
<tr>
<td>Ours (c = 2.0)</td>
<td>88.6</td>
<td>66.1</td>
<td>91.0</td>
<td>81.7</td>
<td>92.7</td>
<td>85.1</td>
<td>50.0</td>
<td>88.9</td>
<td>80.4</td>
<td>7 + 13</td>
<td>20 (20.2%)</td>
</tr>
<tr>
<td>Ours (c = 3.0)</td>
<td>86.4</td>
<td>66.1</td>
<td>91.0</td>
<td>80.7</td>
<td>90.7</td>
<td>85.1</td>
<td>50.0</td>
<td>88.3</td>
<td>79.9</td>
<td>7 + 11</td>
<td>18 (18.8%)</td>
</tr>
<tr>
<td>Ours (w/o KD)</td>
<td>91.7</td>
<td>71.5</td>
<td>91.1</td>
<td>84.5</td>
<td>93.1</td>
<td>89.7</td>
<td>55.2</td>
<td>88.9</td>
<td>83.2</td>
<td>7 + 60</td>
<td>67 (69.8%)</td>
</tr>
</tbody>
</table>

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 $c$. 