Parameter-Efficient Tuning with Special Token Adaptation

Xiaocong Yang†∗, James Y. Huang‡, Wenxuan Zhou‡ and Muhao Chen‡
†Tsinghua University; ‡University of Southern California
yangxc.18@sem.tsinghua.edu.cn;
{huangjam,zhouwenx,muhaoche}@usc.edu

Abstract

Parameter-efficient tuning aims at updating only a small subset of parameters when adapting a pretrained model to downstream tasks. In this work, we introduce PASTA, in which we only modify the special token representations (e.g., [SEP] and [CLS] in BERT) before the self-attention module at each layer in Transformer-based models. PASTA achieves comparable performance to full finetuning in natural language understanding tasks including text classification and NER with up to only 0.029% of total parameters trained. Our work not only provides a simple yet effective way of parameter-efficient tuning, which has a wide range of practical applications when deploying finetuned models for multiple tasks, but also demonstrates the pivotal role of special tokens in pretrained language models.¹

1 Introduction

Built upon a pretrained language model (PLM; Devlin et al. 2019; Liu et al. 2019; Yang et al. 2019; Chowdhery et al. 2022), many of the recent NLP systems are developed based on task-specific fine-tuning. In this way, the PLM effectively leverages the task-agnostic knowledge captured during self-supervised pretraining and adapts itself to downstream tasks. However, full finetuning poses a challenge to model deployment under multi-task, memory-limited scenarios, where we need to train and store a separate full-sized model for each substantially distinct task. As an alternative, parameter-efficient tuning (Ding et al., 2022) aims at only updating a small number of parameters when adapting PLMs to downstream tasks while making most of the model parameters fixed and shared among tasks, thus reducing memory usage.

In this paper, we propose Parameter-efficient tuning with Special Token Adaptation (PASTA), where we only add trainable vectors to hidden representations of special tokens ² at each layer before the multi-head attention module in Transformer-based PLMs. Our work is motivated by the role of special tokens in PLMs. First, special tokens such as [CLS] collect information from the whole input sequence and are typically regarded as the global text representation (Devlin et al., 2019). For sentence-level tasks such as GLUE (Wang et al., 2018), a common practice is to add a new classifier head based on the [CLS] representation in the last model layer. Thus, if trained properly, by updating the [CLS] representations, we can approximate the result of the information collection process in PLMs. Second, many attention heads in PLMs follow a vertical pattern³, where the attention scores are mostly allocated to either the [CLS] or [SEP] token (Clark et al., 2019; Kovaleva et al., 2019), as

Figure 1: Examples of vertical attention heads in the 5-th and 20-th layer of BERT-large with a random sample from CoLA (Warstadt et al., 2019) as input. Heads in the first row and second row assign most of maximal attention weights to [CLS] and [SEP] respectively. See Appx. §C for the full attention map.

¹WLOG, we use the notation of special tokens [CLS] and [SEP] in BERT for the convenience of expression, while the method applies in the same way to other paradigms such as <S> and </S> in RoBERTa (Liu et al., 2019).
²Following Voita et al. (2019) and Yao et al. (2021), an attention head is regarded as vertical if at least 90% tokens assign maximal attention scores to either [CLS] or [SEP].
illustrated in Fig. 1. Therefore, updates to special tokens can also be disseminated to other tokens during the forward pass through the vertical attention heads (Elhage et al., 2021), enabling the PLMs to adapt to both sentential and lexical tasks.

By tuning as few as up to 0.029% of the total parameters, PASTA achieves competitive performance on par with full finetuning and BitFit (Zaken et al., 2022) on GLUE (§4.2). It also outperforms P-tuning v2 (Liu et al., 2022) by 0.6% on CoNLL2003 with 20× fewer additional parameters (§4.3). The ablation study shows that we can further reduce trainable parameters to 0.009% with only a slight performance drop (§4.4), showing the merit of adapting special token representations.

2 Related Work

A recent survey (Ding et al., 2022) categorizes three types of parameter-efficient tuning methods. Addition methods (Houlsby et al., 2019; Lester et al., 2021; Liu et al., 2022) introduce a small number of additional trainable parameters while keeping those in the PLM unchanged. Specification methods (Zaken et al., 2022; Guo et al., 2021; Zhao et al., 2020) update a portion of parameters in the PLM while keeping others frozen. Reparameterization methods (Aghajanyan et al., 2021; Hu et al., 2021; Qin et al., 2021) modify PLMs’ structures to parameter-efficient forms. Our method belongs to the addition-based methods and follows the basic settings of P-tuning v2 (Liu et al., 2022), where newly initialized hidden representations of tokens are inserted into each Transformer layer. Different from most prompt tuning methods that introduce new tokens, we add the introduced vectors to the hidden states of special tokens and keep the sequence length unchanged.

Previous works use probing tasks (Wu et al., 2020) and pruning methods (Prasanna et al., 2020) to study the roles of different modules inside BERT. It has been shown that functional specialization exists in BERT self-attention heads (Clark et al., 2019), and vertical attention heads\(^3\) take up a large portion (Yao et al., 2021). Kovaleva et al. (2019) find that vertical attention heads are almost exclusively associated with attention to [SEP] or [CLS] tokens, and Clark et al. (2019) conclude that heads in early layers often attend to [CLS] while in middle layers attend to [SEP]. In this work, we demonstrate that adapting hidden representations of special tokens is sufficient to bring the performance of PLMs to the level of full finetuning.

3 PASTA

Given a large PLM, our goal is to develop a parameter-efficient tuning method that only updates a small set of parameters when adapting to a downstream task. To this end, we propose a simple yet effective method called PASTA, in which we train a hidden vector for every special token at each Transformer layer, along with a task-specific classifier, while freezing the parameters of the PLM.

3.1 Special Token Adaptation

The special token adaptation is illustrated in Fig. 2. Although these adaptations are not directly applied to non-special tokens, changes in special token hidden states can be effectively disseminated to other tokens via self-attention during forward passes, thanks to the prevalence of vertical attention heads\(^3\) in PLMs.

Specifically, denote the inputs to the \(l\)-th Transformer layer as \(\mathbf{H}\) = \(\{\mathbf{h}_i\}_{i=1}^{N}\), \(\mathbf{h}_i \in \mathbb{R}^d\), where \(N\) is the number of input tokens, \(d\) is the hidden size, PASTA modifies the inputs as follows:

\[
\mathbf{H}_{\text{mod}}^l = \{\mathbf{h}_i + \mathbf{m}_i\}_{i=1}^{N},
\]

\[
\mathbf{H}_{l+1}^l = \text{Trm}^l(\mathbf{H}_{\text{mod}}^l),
\]

where \(\text{Trm}^l\) is the \(l\)-th Transformer layer. \(\mathbf{m}_i \in \mathbb{R}^d\) is our special token adaptation defined as follows:

\[
\mathbf{m}_i = \begin{cases} 
0 & \text{if token } i \text{ is not a special token} \\
e(v_p) & \text{if token } i \text{ is the } p\text{-th special token}
\end{cases}
\]

Figure 2: Architecture of PASTA layer in Transformer. Skip-connections in Transformers are not shown for brevity. At layer \(l\) we add a trainable vector \(e(v_p) \in \mathbb{R}^d\) to the hidden representation of the \(p\)-th special token in the input sequence, and freeze the weights of the PLM.
Table 1: Parameter complexity of PASTA and baselines. Here $L$ and $d$ refer to the number of layers and hidden size of the PLM. $m$ and $r$ refer to the intermediate size of FFN modules in Transformers and Adapters, respectively. $T$ is the prompt length. Parameter consistency refers to whether the set of trainable parameters is consistent across different tasks (Zaken et al., 2022).

<table>
<thead>
<tr>
<th># Param</th>
<th>Parameter Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapter</td>
<td>$O(L \times d \times r)$ ✓</td>
</tr>
<tr>
<td>P-tuning v2</td>
<td>$O(L \times d \times T)$ ✓</td>
</tr>
<tr>
<td>BitFit</td>
<td>$O(L \times (d + m))$ ✓</td>
</tr>
<tr>
<td>Diff-Prune</td>
<td>✓</td>
</tr>
<tr>
<td>PASTA</td>
<td>$O(L \times d)$ ✓</td>
</tr>
</tbody>
</table>

where $e(v^l_p) \in \mathbb{R}^d$ is the trainable vector added to the hidden representation of the $p$-th special token in the input sequence. During downstream task training, only those introduced hidden vectors for special tokens and the task-specific classifier are optimized, and the rest of model parameters are frozen.

### 3.2 Parameter Efficiency and Consistency

As shown in Tab. 1, PASTA achieves $O(L \times d)$ parameter complexity\(^4\) and updates as few as 0.015%-0.029% of the parameters compared to a full PLM when using BERT-large or RoBERTa-large as backbone. Unlike Adapter (Houlsby et al., 2019) that learns the transformation of all input tokens using a shared FFN, PASTA only learns the task-specific update of special token representations as a bias term, which significantly reduces the parameter capacity needed for adaptation on downstream tasks.

Meanwhile, the set of parameters introduced by PASTA is consistent across different tasks, making it efficient for hardware-based deployment (Zaken et al., 2022). On the contrary, in Diff-Prune, the parameter update is considered as a term of the loss function (Guo et al., 2021), resulting in different sets of updated parameters in distinct tasks.

### 4 Experiments and Results

We hereby study the downstream performance of PASTA and analyze the properties of introduced hidden vectors.

#### 4.1 Experimental Setup

**Baseline Methods.** We compare PASTA with the following parameter-efficient tuning methods in prior studies. **Adapter** (Houlsby et al., 2019) introduces new feed-forward modules in Transformer layers while keeping original parameters of the PLM frozen. **BitFit** (Zaken et al., 2022) updates all bias terms in the PLM during finetuning. **Diff-Prune** (Guo et al., 2021) introduces $L_0$-norm penalty on the updated parameters to encourage sparsity of tuned parameters. **P-tuning v2** (Liu et al., 2022) prepends trainable hidden vectors before the input sequence at each layer while keeping the original PLM parameters frozen. **LoRA** (Hu et al., 2021) uses low-rank decomposition matrices to model the parameter updates.

**Model Configuration.** We conduct our experiments using BERT-large and RoBERTa-large (We also report experiments with BERT-base in Appx. §A). To facilitate comparison with baseline works, we take most of the experimental results from their original papers which are reported with either BERT-large or RoBERTa-large. Note that multiple `[SEP]` tokens in a single sequence (e.g., in sentence pair tasks like MNLI) are treated as different special tokens and have separate sets of trainable parameters, and the number of trainable parameters varies among downstream tasks according to the number of special tokens added. Details of training and hyperparameters settings are shown in Appx. §B.

#### 4.2 GLUE Tasks

**Task Setup.** We evaluate PASTA on the widely used GLUE benchmark\(^5\) (Wang et al., 2018). For the convenience of direct comparison, we use the same metrics as were used in baseline works (Devlin et al., 2019; Liu et al., 2019). For experiments with BERT, MRPC and QQP are evaluated using F1 score, STS-B is evaluated using Spearman’s correlation coefficient, CoLA is evaluated using Matthew’s Correlation, and the other tasks are evaluated using accuracy. For experiments with RoBERTa, STS-B is evaluated using Pearson’s correlation coefficient, CoLA is evaluated using Matthew’s Correlation, and the other tasks are evaluated using accuracy.

**Results.** Tabs. 2 and 3 report the performance of PASTA on GLUE benchmark with BERT-large and RoBERTa-large respectively. PASTA with RoBERTa-large achieves the same average score

---

\(4\)The number of special tokens is invariant to the scale of models, and are usually very small.

\(5\)Following previous work (Houlsby et al., 2019; Guo et al., 2021; Zaken et al., 2022), we exclude WNLI since BERT underperforms the majority class baseline (Devlin et al., 2019).
Table 2: BERT-large model performance on GLUE benchmark test set. Lines with * and ** are results from Devlin et al. (2019) and Houlsby et al. (2019), and lines with † and ‡ are from Guo et al. (2021) and Zaken et al. (2022) respectively. We reimplement experiments of P-tuning v2 on GLUE benchmark with a prompt length of 20.

<table>
<thead>
<tr>
<th>Method</th>
<th>% Param</th>
<th>RTE acc.</th>
<th>CoLA mcc.</th>
<th>STS-B Spearman</th>
<th>MRPC acc.</th>
<th>SST-2 F1 acc.</th>
<th>QNLI acc.</th>
<th>MNLI(m/mm) acc.</th>
<th>QQP acc.</th>
<th>Avg. F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Finetuning*</td>
<td>100%</td>
<td>70.1</td>
<td>60.5</td>
<td>86.5</td>
<td>89.3</td>
<td>94.9</td>
<td>92.7</td>
<td>86.7/85.9</td>
<td>72.1</td>
<td>81.6</td>
</tr>
<tr>
<td>Adapter**</td>
<td>3.6%</td>
<td>71.5</td>
<td>59.5</td>
<td>86.9</td>
<td>89.5</td>
<td>94.0</td>
<td>90.7</td>
<td>84.9/85.1</td>
<td>71.8</td>
<td>81.1</td>
</tr>
<tr>
<td>Diff-Prune†</td>
<td>0.5%</td>
<td>70.6</td>
<td>61.1</td>
<td>86.0</td>
<td>89.7</td>
<td>94.1</td>
<td>93.3</td>
<td>86.4/86.0</td>
<td>71.1</td>
<td>81.5</td>
</tr>
<tr>
<td>P-tuning v2</td>
<td>0.29%</td>
<td>70.1</td>
<td>60.1</td>
<td>86.8</td>
<td>88.0</td>
<td>94.6</td>
<td>92.3</td>
<td>85.3/84.9</td>
<td>70.6</td>
<td>81.0</td>
</tr>
<tr>
<td>BitFit‡</td>
<td>0.08%</td>
<td>72.0</td>
<td>59.7</td>
<td>85.5</td>
<td>88.9</td>
<td>94.2</td>
<td>92.0</td>
<td>84.5/84.8</td>
<td>70.5</td>
<td>80.9</td>
</tr>
<tr>
<td>PASTA</td>
<td>0.015%-0.022%</td>
<td>70.8</td>
<td>62.3</td>
<td>86.6</td>
<td>87.9</td>
<td>94.4</td>
<td>92.8</td>
<td>83.4/83.4</td>
<td>68.6</td>
<td>80.9</td>
</tr>
</tbody>
</table>

Table 3: RoBERTa-large model performance on GLUE benchmark. Lines with * are results from Liu et al. (2019), and lines with † are from Hu et al. (2021). We follow the metric settings of baselines and also report results on GLUE development set for the convenience of direct comparison.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Finetuning*</td>
<td>100%</td>
<td>86.6</td>
<td>68.0</td>
<td>92.4</td>
<td>90.9</td>
<td>96.4</td>
<td>94.7</td>
<td>90.2</td>
<td>92.2</td>
<td>88.9</td>
</tr>
<tr>
<td>LoRA†</td>
<td>0.24%</td>
<td>87.4</td>
<td>68.2</td>
<td>92.6</td>
<td>90.9</td>
<td>96.2</td>
<td>94.9</td>
<td>90.6</td>
<td>91.6</td>
<td>89.0</td>
</tr>
<tr>
<td>PASTA</td>
<td>0.015%-0.029%</td>
<td>86.6</td>
<td>69.7</td>
<td>91.8</td>
<td>90.9</td>
<td>96.8</td>
<td>95.1</td>
<td>90.4</td>
<td>89.9</td>
<td>88.9</td>
</tr>
</tbody>
</table>

Table 4: Performance of ablation study with BERT-large on GLUE and CoNLL2003 development sets.

<table>
<thead>
<tr>
<th>Method</th>
<th>CoLA</th>
<th>RTE</th>
<th>MRPC</th>
<th>STS-B</th>
<th>CoNLL2003</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASTA</td>
<td>65.4</td>
<td>76.2</td>
<td>89.7</td>
<td>90.8</td>
<td>94.0</td>
</tr>
<tr>
<td>- w/o [CLS]</td>
<td>58.8</td>
<td>72.6</td>
<td>91.4</td>
<td>90.2</td>
<td>93.7</td>
</tr>
<tr>
<td>- w/o [SEP]</td>
<td>64.5</td>
<td>71.1</td>
<td>91.9</td>
<td>90.3</td>
<td>93.7</td>
</tr>
<tr>
<td>- shared vector</td>
<td>64.7</td>
<td>74.7</td>
<td>92.1</td>
<td>90.0</td>
<td>93.9</td>
</tr>
<tr>
<td>- classifier only</td>
<td>36.5</td>
<td>54.2</td>
<td>81.5</td>
<td>64.9</td>
<td>77.4</td>
</tr>
</tbody>
</table>

Figure 3: NER results on CoNLL03 with BERT-large (F1 score percentages are marked over the bars). Each method is labeled with the percentage of trainable parameter sizes with regard to full finetuning in parentheses.

4.3 Named Entity Recognition

Task Setup. We experiment with the NER task on CoNLL2003 (Tjong Kim Sang and De Meulder, 2003). Following Devlin et al. (2019), we formulate NER as a token classification problem.

Results. As shown in Fig. 3, PASTA with BERT-large achieves an F1 score of 90.8% on CoNLL2003 test set, outperforming P-tuning v2 (Liu et al., 2022) by 0.6% with 20× fewer trainable parameters, while falling behind full finetuning by 2.0%. Nevertheless, the strong performance of PASTA compared to P-tuning v2 indicates that even though PASTA only directly adapts special tokens, the representations of all input tokens can still be properly tuned, supporting our hypothesis that vertical attention heads are able to disseminate adaptations in special token hidden states to other tokens.

4.4 Analysis

Ablation on choices of special tokens. To understand the effect of tuning different combinations of special tokens on downstream tasks, we further limit the additional parameter capacity of PASTA by only adapting either [CLS] or [SEP], or share a common vector across all special tokens. Tab. 4 shows the performance of three ablated variants and a baseline that only tunes the classification head on top of a fixed BERT-large. In general, we observe...
Figure 4: The distribution map of norms of introduced hidden vectors on MRPC, RTE and STS-B tasks with BERT-large. In each subgraph, the first column shows the norms of introduced vectors added to [CLS] at each layer, and the second and third columns are introduced vector norms at two [SEP] tokens respectively.

a decrease in performance on most tasks except on MRPC for three PASTA variants, and performance degrades significantly if we do not adapt any special tokens. These results demonstrate the vital role of introduced hidden vectors for special tokens in PASTA, while the best choice of special tokens to be adapted may vary depending on the task.

Norm distribution of introduced hidden vectors. Fig. 4 shows the norm distribution of introduced vectors on downstream tasks. The introduced hidden vectors learn the difference of special tokens between pretrained and adapted models, and thus norms of those vectors indicate the magnitude of parameter change at different layers. Similar to the pattern of parameter change during full finetuning (Kovaleva et al., 2019), PASTA generally has larger norms of hidden vectors at layers closer to the output.

5 Conclusion

We present PASTA, a parameter-efficient tuning method that only modifies special token representations at each Transformer layer when adapting to downstream tasks. Our approach is motivated by the observation that PLMs have a large amount of vertical attention heads that heavily attend to special tokens, and these heads disseminate value updates from special tokens to all of the other tokens. Experiments show that PASTA achieves strong performance comparable to full finetuning on sentential and lexical tasks with high parameter efficiency. Our work not only provides an effective solution for parameter-efficient tuning, but also demonstrates the pivotal role of special tokens in PLMs.

Limitations

In this work we hypothesize that the vertical attention heads could play a role as “information disseminator” based on the theoretical analysis of Transformers (Elhage et al., 2021). However, we still have no direct approaches such as probing tasks and reverse engineering to prove this assumption. And since PASTA relies on adaptation of special tokens, it cannot be applied to language models which do not pad special tokens to input sequences such as GPT-2 (Radford et al., 2019). For the empirical results, we choose GLUE benchmark and CoNLL2003 to study the performance on language understanding tasks. The effectiveness of PASTA on language generation tasks has not been tested in this work due to limited bandwidth. Finally, similar to other parameter-efficient tuning methods, PASTA suffers from a higher computational cost compared to full finetuning.

Acknowledgement

We appreciate the anonymous reviewers for their insightful comments and suggestions. Xiaocong Yang, Wenxuan Zhou and Muhao Chen are supported by the National Science Foundation of United States Grant IIS 2105329. This material is supported in part by the DARPA MCS program under Contract No. N660011924033 with the United States Office Of Naval Research, an Amazon Research Award, a Cisco Research Award and a subaward from NSF Cloudbank 1925001.

References


Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsyvashchenko, Joshua Maynez, Abhishek


Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madiand Khaba. 2021. Unipelt: A unified framework for parameter-efficient language model tuning.


Yujia Qin, Xiaozhi Wang, Yusheng Su, Yancai Lin, Ning Ding, Jing Yi, Weize Chen, Zhiyuan Liu, Juanzi Li, Lei Hou, Peng Li, Maosong Sun, and Jie Zhou.


Figure 5: Full attention map of BERT-large pretrained with a random sample from CoLA as input. Rows and columns represent model layers and heads respectively, and darker color indicates larger weights. Vertical attention heads are highlighted in orange.

<table>
<thead>
<tr>
<th>%Param</th>
<th>RTE</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>SST-2</th>
<th>QNLI</th>
<th>MNLI (m/mm)</th>
<th>QQP</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Finetuning *</td>
<td>100%</td>
<td>66.4</td>
<td>62.1</td>
<td>89.8</td>
<td>90.9</td>
<td>91.6</td>
<td>90.0</td>
<td>83.2/-</td>
<td>87.4</td>
</tr>
<tr>
<td>Adapter*</td>
<td>0.81%</td>
<td>71.8</td>
<td>61.5</td>
<td>88.6</td>
<td>89.9</td>
<td>91.9</td>
<td>90.6</td>
<td>83.1/-</td>
<td>86.8</td>
</tr>
<tr>
<td>BitFit†</td>
<td>0.8%</td>
<td>72.3</td>
<td>58.8</td>
<td>89.2</td>
<td>90.4</td>
<td>92.1</td>
<td>90.2</td>
<td>81.4/-</td>
<td>84.0</td>
</tr>
</tbody>
</table>

Table 5: PASTA with BERT-base model performance on GLUE benchmark development set. Lines with * and † refer to results from Mao et al. (2021) and Zaken et al. (2022) respectively.

<table>
<thead>
<tr>
<th>RTE</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>SST-2</th>
<th>QNLI</th>
<th>MNLI (m/mm)</th>
<th>QQP</th>
<th>CoNLL2003</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate</td>
<td>4.5e-3</td>
<td>5e-3</td>
<td>2e-3</td>
<td>2.5e-3</td>
<td>7e-3</td>
<td>2e-3</td>
<td>5e-4</td>
<td>3e-3</td>
</tr>
<tr>
<td>Batch size</td>
<td>32×4</td>
<td>32×1</td>
<td>32×3</td>
<td>32×4</td>
<td>64×3</td>
<td>32×4</td>
<td>32×1</td>
<td>32×4</td>
</tr>
<tr>
<td>Number of adapted tokens</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Training epochs</td>
<td>150</td>
<td>100</td>
<td>150</td>
<td>150</td>
<td>100</td>
<td>80</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>Best dev performance</td>
<td>76.2</td>
<td>65.4</td>
<td>90.8</td>
<td>89.7</td>
<td>93.9</td>
<td>92.2</td>
<td>83.7</td>
<td>87.9</td>
</tr>
<tr>
<td>Best epochs</td>
<td>121</td>
<td>63</td>
<td>109</td>
<td>136</td>
<td>99</td>
<td>42</td>
<td>49</td>
<td>97</td>
</tr>
</tbody>
</table>

Table 6: PASTA with BERT-large training details for GLUE and CoNLL2003 tasks. Distributed training on multiple GPUs is used when available for less training time.