Parameter-Efficient Tuning with Information Carrier and Partially Unfrozen Component

Anonymous ACL submission

Abstract

Recent developments in Large Language Models (LLMs) have motivated widespread research interest in exploring their potential in downstream applications. Given the black box nature of LLMs, prompt engineering becomes the natural method to interact with them. Prompt tuning, including hard and soft prompts, combine manually or automatically composed text templates with adjustable vectors, aiming to improve parameter efficiency with only a small fraction of tunable parameters. However, the training cost has not been significantly reduced due to the presence of network-wide backpropagation, and it often leads to moderate performance deterioration with a soft prompt initialization issue. Late Prompt Tuning (LPT) further reduces training cost and performance drop, by inserting a parameterized vector into the center of the model and introducing intrinsic dimension into the initial soft prompt. With significant saving in training cost, performance of LPT is still weak compared to other parameter-efficiency methods like adapter and LoRA (Low-Rank-Adaptation). We argue that it is caused by the limited capacity of soft prompts to carry complete downstream task information. To deal with this issue, we propose a new parameter-efficient tuning method called Back-and-Forth Tuning (BFT), which achieves better results by combining hard prompts and task information. With a new information component and partially unfrozen module, our model can store task-specific information with limited parameters. Soft prompt and adapter are both viable options for the information component, and results show better performance of the latter. Comprehensive experiments demonstrate that our method improves 2.27% over LPT on average accuracy of 10 tasks, together with faster convergence speed and no increase in training cost.

1 Introduction

With the widespread application of the Transformer (Vaswani et al., 2017) framework, pre-trained large language models have become the undoubtful cornerstone of the Natural Language Processing (NLP) research field. Since the proposal of BERT (Devlin et al., 2018), pre-trained models (Devlin et al., 2018; Lewis et al., 2019; Liu et al., 2019; Raffel et al., 2020; Brown et al., 2020; He et al., 2020; Joshi et al., 2020; He et al., 2021b) have been constantly refreshing benchmarks in NLP with updates on the transformer network. Moreover, (Kaplan et al., 2020) discovered that larger models are significantly more sample-efficient with their stronger representative power.

Full parameter fine-tuning of a model is a popular method to adapt to downstream tasks. However, as the number of parameters in a model increases, it becomes more and more costly to fine-tune all of them. Parameter-efficient fine-tuning is a new research area that tries to adjust a pre-trained model to downstream tasks by fine-tuning only part of the model's parameters.

The concept of parameter-efficient fine-tuning was first introduced in the adapter framework (Houlsby et al., 2019). It only adjusts a small percentage of the model's parameters, ranging from 0.5% to 8%, while achieving competitive results. Other parameter-efficient fine-tuning methods, such as soft prompt (Lester et al., 2021), Bitfit (Ben Zaken et al., 2022), prefix tuning (Li and Liang, 2021), and LoRA (Hu et al., 2021), also aim to tune a minimal number of parameters to reduce costs.

However, there is a common misconception that reducing the number of adjustable parameters directly translates to cost reduction. As shown in Figure 1, there is no direct relationship between free parameters and memory requirement. Although the methods above have reduced the number of tunable parameters to about 1% of the original model, they only result in a reduction of approximately 20% in GPU memory usage and a modest speedup of about 1.5 times in training efficiency. This limited

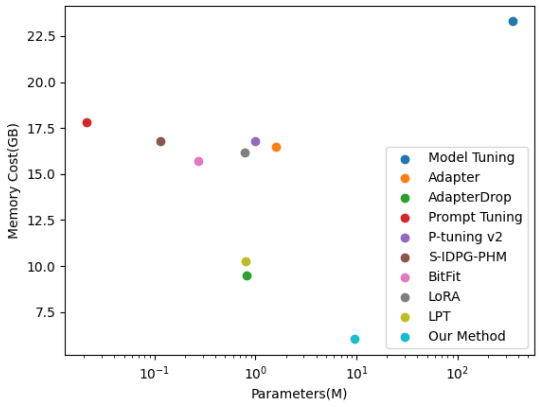


Figure 1: Number of free parameters and required memory for multiple parameter efficiency methods when applied to RoBERTa-Large model.

improvement is attributed to the impact of backpropagation. Most parameters, including the fixed ones, still need to go through the backpropagation process. It greatly reduces the expected improvement in efficiency.

In contrast, Late Prompt Tuning (LPT) (Liu et al., 2022b) proposed a solution by embedding a soft prompt into a hidden layer of the model which is closer to the output. Through carefully-designed experiments, the researchers found that embedding the soft prompt in the middle of the model's hidden layer yields the best result. This method effectively reduces the backpropagation distance by half, leading to significant reductions in training time and GPU memory usage. Specifically, LPT achieves 2.0× training speed of and reduces memory costs by 56.6% compared to a full model tuning on RoBERTa-Large (Liu et al., 2019).

In this paper, our objective is to continue improving over the convergence speed of LPT and further reduce training time. Inserting the soft prompt into the hidden layer in the middle of the model would face the same issue of transmission distance as LPT. Experiments prove that this issue has negative impacts the convergence speed as well as model accuracy. To address it, we also modified the parameters of the last layer to better align with the soft prompts in the middle of the model.

Building upon co-tuning of the middle and last layers, we propose a method called Back-and-Forth Tuning(BFT). Since the design is based on LPT with additional parameters behind its hidden layer, the memory footprint and training time stay about the same as the original approach. Extensive experiments demonstrate that compared with the traditional LPT under the RoBERTa-Large model, BFT converges faster and leads to about 1.99% accuracy improvement, with similar memory usage and training speed. In addition, we also explore the feasibility of inserting a single hidden layer inside the model in similar methods like adapter, resulting in a better performing architecture.

In summary, main contributions of this paper are as follows:

1. We explore the mechanism behind the cost-effectiveness of LPT training and introduce further architectural modifications.

2. Combining the soft prompt and partially unfrozen modules, we propose BFT as an improvement over LPT, showing faster convergence and better performance without increasing cost.

3. With an adapter-based information component added to the BFT framework, the BFT-adapter model achieves even better performance. In comparison to LPT under the RoBERTa-Large model, it reduces training time by 16.5% and training cost by 41.4%, and average accuracy is 2.27% higher in 10 tasks.

2 Related Work

Parameter-Efficient Fine-Tuning (PEFT) methods mainly focus on reducing the number of tunable parameters. They are mainly divided into four categories.

Addition-based PEFT: All the original parameters in the network are frozen, with only additional parameters fine-tuned. According to where the additional parameters are added, it includes two categories: adding parameters at the architecture level and to the existing vectors. For the first type, Adapter (Houlsby et al., 2019) add a bottleneck network twice to each transformer layer: after the projection following multi-head attention and after the two feed-forward layers, while tiny attention adapter (Zhao et al., 2022) adds a small attention mechanism (Vaswani et al., 2017) after the transformer layer. Although models of this type perform well in full-sample learning, they tend to yield lower performance in few-shot learning. As for the second type, soft prompt (Lester et al., 2021) adds additional parameters to input embeddings, prefix tuning (Li and Liang, 2021) links them to

the key and value matrices in the attention mechanism, P-Tuning v2 (Liu et al., 2021) injects soft prompts into each layer of the model's hidden layer for better performance, LPT (Liu et al., 2022b) only connects parameters to hidden vectors in the middle layer of the model, and $(IA)^3$ (Liu et al., 2022a) adds parameters to the key and value matrices in the attention mechanism in the vertical dimension plus the feedforward mechanism in the parallel dimension. Except for soft prompt and LPT that only make modifications to one specific layer, other models have additional parameters added to each layer.

Specification-based PEFT: In this method, most of the original parameters are frozen, and only the specified parameters are fine-tuned. For example, Bitfit (Zaken et al., 2021) fine-tunes only the biases of the model, and (Lee et al., 2019) achieves 90% performance of the original model by fine-tuning only the last few layers.

Re-parameterization PEFT: This method keeps the original parameters but re-parameterizes the update vectors. For instance, (Aghajanyan et al., 2020) replaces the update of the original parameters with the product of two low-rank matrices, and LoRA (Hu et al., 2021) re-parameterizes only the query and value matrices in the attention mechanism at each layer of the model.

Fusion method: This type of approaches combines multiple methods mentioned above. Compacter (Karimi Mahabadi et al., 2021) re-parameterizes the matrix of the Adapter architecture and shares a subset of parameters in each layer of the model. (He et al., 2021a) combines Adapters with prefix tuning and LoRA to explore a better model, and they find that parallel adapters provide the best performance. Delta tuning (Ding et al., 2022) combines the aforementioned methods and designs an automated process to select the appropriate combination for the tasks at hand. Black box tuning (Sun et al., 2022) re-parameterizes and de-gradients the updates of soft prompts, providing an idea for personalized fine-tuning of large models like GPT-3 (Brown et al., 2020).

3 Method

3.1 PEFT revisited

The original intention of PEFT was to reduce both the storage cost and the training cost associated with each downstream task. However, traditional focus of the field has shifted towards minimizing the number of adjustable parameters, primarily addressing the storage problem without considering the training cost. Our research aims to find a balance between the space and time factors and evaluate the results accordingly.

The number of adjustable parameters in PEFT methods is usually small, but the training cost has not been significantly reduced, possibly due to the influence of wide-range backpropagation. From our understanding, the PEFT approach resembles a variant of Prompt cues. It fuses additional information about downstream tasks into these parameters by adjusting a small portion of the model parameters. Depending on specific parameters adjusted, the information carries different types of capabilities that will empower the model for solving corresponding tasks.

The optimal placement of prompt information has been a key consideration in the evolution of prompt engineering. Initially, manually designed hard prompts were employed to simulate training data related to the downstream task. Due to variations in manually designed templates and significant disparities in results, some researchers shifted towards soft prompts. Original soft prompts were placed at the input layer, but they tend to yield unsatisfactory outcomes. P-Tuning v2 (Liu et al., 2021) introduced prompts in each hidden layer of the model, resulting in inspiring results, but the training cost was also comparable to a full parameter tuning. LPT places soft prompts in a single layer, which yields similar results to P-Tuning v2 with significant advantages in training cost and parameter quantity. We follow the LPT approach and further explore possible improvements in convergence speed and results.

3.2 BFT: Back-and-Forth Tuning

Problem Definition. Given a pre-trained language model **M**, we begin by reconstructing the sentence S using a hard prompt generator **P**, resulting in **E**([CLS] **P**(**S**) [SEP]), where **E** represents the embedding layer of model **M**. **P** has distinct designs for various downstream tasks, each incorporating a [MASK] token. Subsequently, we inject **E** into the pre-trained model, and hidden state of the [MASK] token in the last hidden layer is employed to predict the label **Y** using a language model head. Notably, **Y** is part of a small dictionary of model **M** we have defined.

Architecture. Based on the previous analysis, the

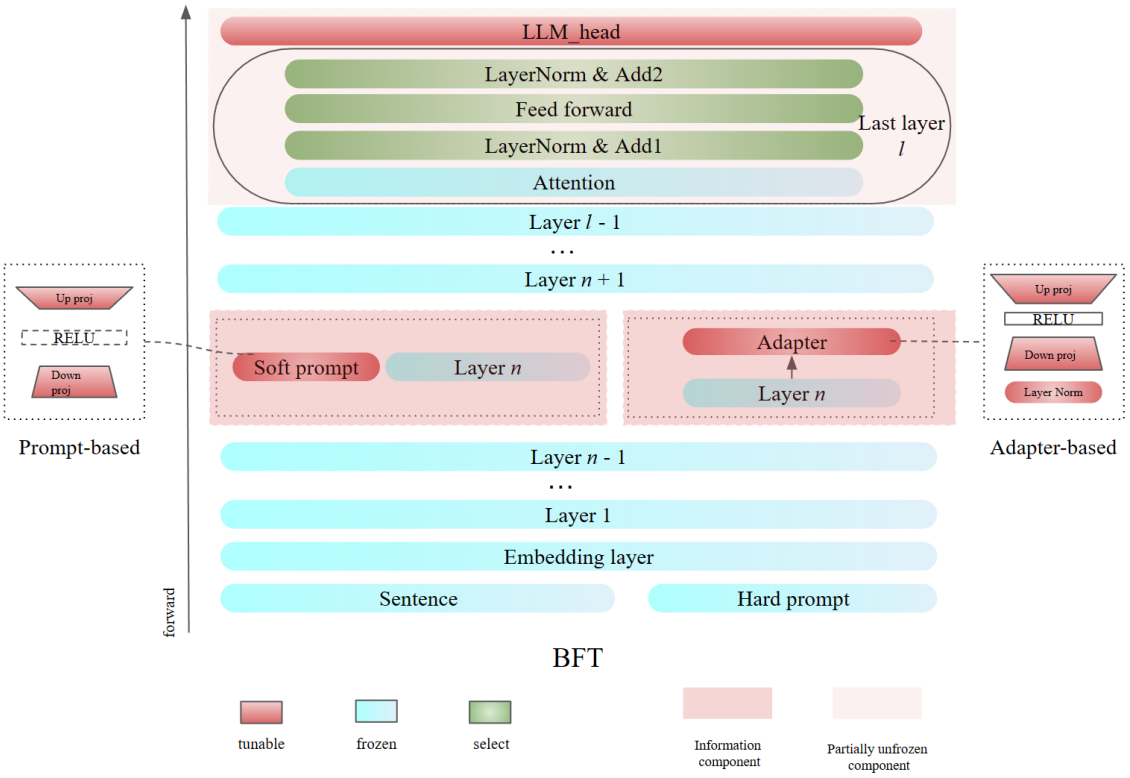


Figure 2: An illustration of BFT. Red color represents modules with adjustable parameters. Green modules may have gradients optionally turn on. It should be noted that when the gradient in a module is turned on, the modules above it also enter the gradient backpropagation process. According to the different information component, the model is divided into BFT and BFT-adapter. The dotted box in soft prompt indicates its optional RELU module.

architecture we have ultimately chosen is illustrated in Figure 2. It is mainly composed of three parts: hard prompt, information component, and partially unfrozen component. Only parameters of the information module and the partpartially unfrozen are fine-tuned, with other parameters frozen. Among them, hard prompt is used to shorten the gap between downstream task and pre-training, information component is used to store downstream task information, and partially unfrozen component is used to coordinate fine-tuning with information component. It should be noted that the information component has two types of different architecture. One is soft prompt that forms the BFT model. The other is BFT-adapter model which has an adapter module in its place. The partpartially unfrozen has a different variants Depending on the internal architecture of the information component, the specified parameter module also has different setup.

3.2.1 hard prompt

Combination of soft and hard prompts has shown improved results (Liu et al., 2021; Han et al.; Hambardzumyan et al., 2021; Zhong et al., 2021). All the tasks we selected are classification tasks, including single-sentence classification and multi-sentence classification. For the single-sentence classification tasks, the input sentence is directly fed into the hard prompt. A single-sentence classification prompt is "It was <mask>. ", where <mask> is the verbalizer token limited to a small vocabulary.

Regarding multi-sentence classification tasks, the approach is similar to single-sentence classification. Due to contextual considerations, the position of the prompt is not fixed at the end. Multiple hard prompt formats are considered for different tasks. This means that hard prompts can be placed between two sentences, before sentences, or after sentences. See the Appendix A for more descriptions of the hard prompts.

3.2.2 information component

For the soft prompt from BFT, we follow the approach of (Wang et al., 2021) and utilize the mutual information between the hidden states and label of each input. Regarding the initialization of soft prompts, (Gu et al., 2021) have explored this issue.

Due to cost and time constraints, we did not employ the pre-training method for soft prompt mentioned by (Gu et al., 2021). Instead, we adopted one of their conclusions, which suggests that random initialization performs better than initialization from a dictionary or the last classification label.

For the architecture of soft prompts, we employ a re-parameterization approach. This involves transforming the soft prompt matrix into the product of two smaller matrices. There are two re-parameterization methods available: traditional feed forward or direct matrix multiplication. Taking the former approach, the soft prompt **SP** can be represented by the following formula:

$$\mathrm{SP}_1 = \mathrm{W}_2(\mathrm{RELU}(\mathrm{W}_1 \times \mathrm{h_n} + \mathrm{b}_1)) + \mathrm{b}_2 \quad (1)$$

$$\mathrm{SP} = \mathrm{reshape}(\mathrm{SP}_1) \quad (2)$$

where $W_1 \in R^{m\times d}, W_2 \in R^{(l\times d)\times m}, h_n \in R^d$ and $SP \in R^{l\times d}$. In the matrices, m represents the lower dimension of our projection, d represents the dimension of the hidden layer, and h_n represents the n-th hidden layer from which we specify the added information module from the model. The range of n depends on the backbone model we choose. In RoBERTa-Large model which has 24 layers, the range of n is 1 to 24. The re-parameterization effect becomes apparent when d is significantly larger than m.

For the adapter (Houlsby et al., 2019) from BFT-adapter, we take the original bottleneck architecture and attach it behind the hidden layer. Its detailed formula is shown below:

$$\mathrm{h_{tmp}} = \mathrm{W}_1 \times \mathrm{LayerNorm}(\mathrm{h_n}) + \mathrm{b}_1 \quad (3)$$

$$\mathrm{Adapter}(\mathrm{h_n}) = \mathrm{W}_2 \times \mathrm{RELU}(\mathrm{h_{tmp}}) + \mathrm{b}_2 \quad (4)$$

where $W_1 \in R^{m\times d}, W_2 \in R^{(l\times d)\times m}$ and $h_n \in R^d$. In the matrices, m represents the lower dimension of our projection, d represents the dimension of the hidden layer, and h represents the hidden layer. The new hidden layer h_{new} that leads to the next layer is as follows:

$$\mathrm{h_{new}} = \mathrm{Adapter}(h_n) + h_n \quad (5)$$

Here, h_{new} indicates that after the n-th layer hidden layer is updated, it still leads to the original n+1 layer model.

3.2.3 partially unfrozen component

To lower training cost in LLMs, LPT reduces the distance of backpropagation for soft prompts, but its improvement is limited from the performance perspective. Taking RoBERTa-Large as an example, when the soft prompts are placed in the hidden layer in the middle of the model, specifically in the 12-th layer, all 12 layers leading up to the final output have to be fully back-propagated to get to the soft prompt. LPT also demonstrates that placing the soft prompts closer to the output leads to degraded results. Long-distance transmission becomes its performance bottleneck. In order to reduce training cost, we focus on the last few layers with the idea in (Lee et al., 2019). In addition, we look for other architectures that would allow the component to be placed closer to the output of the model, further reducing cost.

The partially unfrozen component selects certain parameters located in the last layers. Gradients of these parameters are turned on to coordinate training with the information module. For cost considerations, we set the scope of the selected parameters at LLM_head and the Encoder at the last layer of the language model. The detailed experiment is carried out in Section A.5, and three variants are finally selected: The BFT benchmark method which turns on the gradient only in the LLM_head module; BFT with ffd variant method which opens the gradient of all parameters from the feed forward module to the output module; BFT with ln which opens the gradient for all parameters from the layer normalization (Ba et al., 2016) module up to the output. From Figure 2, the layer normalization module is right after attention.

4 Experiments

4.1 Datasets

For a fair comparison to LPT, we use the same datasets including 5 single-sentence classification and 5 multi-sentence classification tasks. Six of the datasets are from GLUE (Wang et al., 2018), and the other four include MR (Pang and Lee, 2005), SUBJ (Pang and Lee, 2005), TREC (Voorhees and Tice, 2000) and MPQA (Wiebe et al., 2005). See the Appendix A for more details about datasets.

4.2 Experiment Settings

All experiments were run on a single NVIDIA RTX-3090 GPU with 24GB memory. The RoBERTa-Large model (Raffel et al., 2020) is

taken as the backbone and tested on full task data. Most of our hyperparameters follow the design of LPT, with an exception of prompt length, which is optimized for our model. The adapter information module is inserted into layer 18 and the prompt information module is inserted into layer 12, which is derived from the results of section 4.4.1 on two datasets. For the partially unfrozen component, BFT is the base method and the two variants models from Table 7 are selected: BFT with ffd and BFT with ln. Regarding the length of the prompt information module, we set it to 5 in all tasks. The loss curve was observed on the GPT-2 model (Radford et al., 2019), with an insertion position located at the 18-th layer. All datasets use 10 epochs. More implementation details are provided in the Appendix A.

4.3 Baseline

Baseline methods include LPT and the earlier models in Section 2. Addition-based PEFT methods include Adapter (Houlsby et al., 2019), Adapter Drop (Rücklé et al., 2020), Prompt Tuning (Lester et al., 2021), P-tuning v2 (Liu et al., 2021) and IDPG (Wu et al., 2022). The only specification-based methods is Bitfit (Zaken et al., 2021). For Re-parameterization-based PEFT based methods, we selected LoRA (Hu et al., 2021). Results are taken directly from the LPT paper (Liu et al., 2022b), with an exception of LPT, whose performance cannot be accurately replicated in our experiment. Except for the results taken directly from reference, three random seeds were used for all results. BFT is the prompt-based method, also referred to as LLM_head in Table 1. BFT with LN and BFT with FFD represent ADD&NORM1 and FFD in Table 1, respectively. BFT-adapter is the adapter-based method.

4.4 Results

4.4.1 Memory profile

In this section, we mainly explore correlation between the use of GPU memory and the number of adjustable parameters. Table 1 shows the memory footprint and training speedup for various PFET methods. As we can see, excessive attention on reducing parameter count may not result in comparable reduction in training costs. For example, IDPG and prompt tuning methods show smallest number of parameters within these methods, but their training speed and memory saving are almost worst. On contrary, LPT keeps a much larger number of adjustable parameters, but the training speed up and memory usage are one of the best.

[1] https://github.com/xyltt/LPT

Method	Tuable Parameters	Training Speed tokens/ms(↑)	Memory Cost GB(↓)
Model Tuning	355M	11. 6	23. 3
Adapter	1. 6M	15. 5 (1. 3×)	16. 5 (29. 8%)
AdapterDrop	811K	21. 6 (1. 9×)	9. 5 (59. 6%)
Prompt Tuning	21K	16. 9 (1. 5×)	17. 8 (24. 3%)
P-tuning v2	985K	19. 2 (1. 7×)	16. 8 (28. 5%)
S-IDPG-PHM	114K	12. 0 (1. 0×)	16. 8 (28. 5%)
BitFit	273K	16. 5 (1. 4×)	15. 7 (33. 2%)
LoRA	788K	16. 4 (1. 4×)	16. 2 (31. 1%)
LPT	792K	23. 2 (2. 0×)	10. 285(55. 9%)
BFT	1. 9M	23. 2 (2. 0×)	10. 287(55. 8%)
BFT with ffd	10. 289M	23. 2 (2. 0×)	10. 407(55. 3%)
LPT-adapter	36K	**27. 8(2. 4×)**	**5. 949(74. 5%)**
BFT-adapter	1. 13M	27. 8 (2. 4×)	5. 999(74. 3%)
BFT-adapter with ln	9. 535M	27. 8 (2. 4×)	6. 027(74. 1%)

Table 1: Full data comparison on ten classification tasks. All results were obtained under the test set except for the six datasets of GLUE. LPT results are replicated from its codebase[1]. Results for other methods are taken directly from the LPT paper. We report mean and standard deviation of performance over 3 different random seeds for LPT and BFT. All the results are obtained using RoBERTa-Large. Bold face shows the best result for each column, and the underline score is the second best.

With this important observation, we argue that the main training cost are involved in error back-propagation instead of parameter updates. Therefore, some PFET methods may not effectively reduce training costs even with a small number of parameters. Our research direction focuses on reducing backpropagation distance or adopting more efficient parameter update method, such as derivative-free optimization (Hansen and Ostermeier, 2001; Hansen et al., 2003; Rios and Sahinidis, 2013). In this paper, we mainly use the former approach to find a better trade-off in terms of training budget and task performance. As can be seen from Table 1, our LPT-based improvements do not involve significantly performance cost even with additional parameters, while staying far ahead of other models in case of training speed and memory costs.

4.4.2 Main Results

Main experiment results are shown in Table 2. On the average of 10 tasks, BFT has clear performance improvement over traditional LPT method. Especially for the four tasks with test datasets (MPQA, MR, Subj, TREC), BFT significantly improves over LPT. Compared with the original LPT, LPT-adapter variant has obvious advantages over the prompt architecture on most tasks expect SST-2. An interesting observation is that prompt-based

Method	Tunable Parameters	SST-2 (acc)	MPQA (acc)	MR (acc)	Subj (acc)	TREC (acc)	MNLI (acc)	MRPC (acc and F1)	QNLI (acc)	QQP (acc and F1)	RTE (acc)	Avg (acc)
Model Tuning	355M	95. 6	90. 2	91. 3	96. 8	97. 6	89. 3	91. 2	94. 6	90. 7	86. 2	92. 35
Adapter	1. 6M	**96.2** (0.2)	89. 2 (0. 5)	91. 6 (0. 4)	96.8 (0.4)	97. 0 (0. 3)	**90. 5**(0. 1)	90. 3 (1. 0)	94. 7(0. 3)	89. 4(0. 7)	**85. 5**(1. 2)	**92. 3**
AdapterDrop	811K	95. 3 (0. 3)	89. 1 (0. 7)	91. 0 (0. 5)	95. 3 (0. 6)	95. 7 (0. 5)	88. 5 (0. 2)	90. 1 (1. 3)	93. 3 (0. 3)	88. 3 (0. 3)	81. 1 (2. 0)	90. 8
Prompt Tuning	21K	94. 9 (0. 5)	88. 8 (0. 8)	89. 6 (0. 5)	93. 9 (0. 6)	86. 4 (0. 7)	86. 7 (0. 9)	75. 7 (0. 7)	91. 4 (0. 1)	81. 2 (0. 8)	60. 8 (0. 5)	84. 9
P-tuning v2	985K	95. 8 (0. 4)	89. 9 (0. 6)	91. 4 (0. 4)	96. 5 (0. 2)	95. 8 (0. 6)	88. 2 (0. 2)	86. 5 (2. 1)	93. 7 (0. 3)	85. 3 (0. 2)	66. 9 (2. 3)	89
S-IDPG-PHM	114K	94. 8 (0. 3)	89. 5 (0. 6)	90. 8 (0. 5)	95. 9 (0. 6)	89. 3 (0. 4)	87. 4 (0. 5)	77. 3 (1. 2)	91. 2 (0. 4)	82. 3 (1. 9)	62. 7 (1. 9)	86. 1
BitFit	273K	95. 9 (0. 1)	89. 2 (0. 9)	91. 8 (0. 5)	96. 9 (0. 1)	96. 2 (0. 3)	90. 0(0. 1)	89. 6 (0. 9)	94. 4 (0. 2)	87. 9 (0. 4)	82. 4 (1. 1)	91. 4
LoRA	788K	96. 2 (0. 3)	90. 1 (0. 3)	92. 0 (0. 1)	97. 1(0. 4)	96. 8 (0. 6)	89. 8 (0. 3)	**91. 1** (0. 6)	**94. 8**(0. 2)	**89. 8** (0. 1)	84. 8 (2. 1)	**92. 3**
LPT	792K	95. 68 (0. 46)	90. 89 (0. 26)	91. 23 (0. 15)	96. 23 (0. 14)	94. 67 (0. 5)	86. 50 (0. 16)	83. 79 (1. 2)	90. 47 (0. 49)	83. 77 (0. 09)	76. 17 (0. 72)	88. 98
BFT	1. 9M	95. 53 (0. 11)	91. 83 (0. 4)	92. 19 (0. 15)	96. 57 (0. 44)	96. 47 (0. 42)	88. 31 (0. 06)	82. 91 (0. 98)	91. 81 (0. 04)	87. 64 (0. 05)	80. 02 (0. 55)	90. 43
BFT with ffd	10. 289M	95. 18(0. 2)	**96. 9** (0. 43)	93. 51(0. 44)	96. 58 (0. 08)	96. 33(0. 61)	88. 33(0. 1)	85. 31(1. 49)	92. 18(0. 5)	87. 83(0. 03)	77. 5(1. 71)	90. 97
BFT with ln	10. 291M	95. 49(0. 47)	95. 9(1. 04)	93. 61(0. 43)	96. 53 (0. 13)	96. 33(0. 61)	88. 19(0. 09)	84. 51(1. 55)	91. 62(0. 72)	88. 2(0. 09)	75. 93(1. 78)	90. 63
LPT-dapter	36K	94. 99(0. 52)	92. 18(0. 3)	92. 52(0. 24)	96. 64(0. 3)	**97. 0**(0. 2)	86. 93(0. 16)	83. 72(0. 04)	91(0. 05)	86. 01(0. 06)	77. 98(0. 36)	89. 9
BFT-adapter	1. 13M	95. 14(0. 73)	92. 92(0. 08)	93. 59(0. 25)	96. 98(0. 16)	96. 67(0. 12)	87. 57(0. 1)	86. 05(0. 3)	91. 24(0. 09)	87. 57(0. 1)	78. 22(0. 75)	90. 64
BFT-adapter with ffd	9. 533M	94. 92(0. 35)	95. 94(0. 51)	**94. 81**(0. 47)	97. 03(0. 19)	96. 67(0. 42)	87. 48(0. 09)	87. 48(1. 2)	91. 59(0. 22)	87. 48(0. 09)	76. 41(1. 99)	91. 08
BFT-adapter with ln	9. 535M	95. 07(0. 23)	95. 89(0. 35)	94. 8(0. 36)	**97. 2**(0. 1)	96. 8(0. 2)	87. 52(0. 07)	88. 15(1. 27)	91. 29(0. 08)	87. 52(0. 07)	77. 14(1. 24)	91. 25

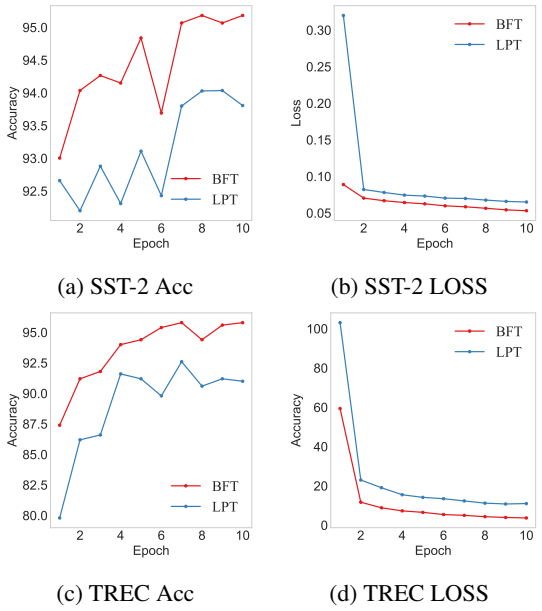
Table 2: Full data comparison on ten classification tasks. All results were obtained under the test set except for the six datasets of GLUE. LPT results are replicated from its codebase. Results for other methods are taken directly from the LPT paper. We report mean and standard deviation of performance over 3 different random seeds for LPT and BFT. All the results are obtained using RoBERTa-Large. Bold face shows the best result for each column except for model tuning, and the underline score is the second best.

methods such as P-tuning v2 and prompt tuning in the table is generally not as good as the adapter-based methods such as Adapter drop and Adapter. The same applies to our proposed new architecture, BFT-Adapter. Looking closely at the result distribution, it can be found that prompt methods have clear disadvantage in the tasks of MNLI, MRPC, QNLI, QQB, RTE. These are all inference tasks and Similarity and Paraphrase tasks belonging to the GLUE dataset. We speculate the prompt methods can only carry superficial task information, so their performance is on par with the adapter-based methods in simple single-sentence tasks, failing the more complex tests.

The traditional LPT method only injects one layer of prompt information, and its limited capacity results in poor performance in complex tasks. With the additional adjustable parameters in the BFT model, it can alleviate the above problems without increasing the cost. In the prompt-based architecture, results get better as the number of parameters increases. The '*BFT with ffd*' model has performace comparable to some low-cost adapter architecture models such as AdapterDrop (Rücklé et al., 2020). Although the adapter carries more information than a single prompt, single layer, still limits the amount of information can carry. Therefore, increasing the number of adjustable parameters is also effective for LPT-Adapter. Unlike the prompt-based method, the adapter-based method has a rebound effect. Adding one more layer of gradient propagation to '*BFT-adapter with ffd*' does not show significant performance improvement, so our assumption is that involving more layers may result in overfit.

A more interestingly observation is that MR and MPQA tasks benefit fromt the additional parameters, no matter the base model is a prompt-based or adapter-based method. The parameter unfreezing process can indeed improve the information acquisition ability of the LPT model. For example, on the MPQA and MR data sets, some models with unfrozen modules have helped the benchmark model reach another level.

5 Ablation Study



(a) SST-2 Acc (b) SST-2 LOSS

(c) TREC Acc (d) TREC LOSS

Figure 3: Learning curve showing training loss and accuracy of GPT-2 model on SST-2 and TREC.

5.1 Convergence speed

In addition to final performance, training curve is another factor to consider. To compare the conver-

Method	Tunable Parameters	SST-2 (acc)	MPQA (acc)	MR (acc)	Subj (acc)	TREC (acc)	MNLI (acc)	MRPC (acc and F1)	QNLI (acc)	QQP (acc and F1)	RTE (acc)	Avg (acc)
5epoch												
LPT	792K	94. 6	90. 8	90. 9	95. 6	84	86. 3	82. 17	90. 5	83. 4	75. 8	87. 41
BFT	1. 9M	94. 2	92. 2	91. 4	**97. 2**	95. 8	**88. 1**	82. 21	86. 9	86. 4	**78**	89. 24
LPT-adapter	36K	**94. 95**	92. 05	92. 23	96. 65	**97**	86. 22	82. 4	90. 88	85. 23	75. 45	89. 31
BFT-adapter	1. 13M	94. 27	**92. 6**	**92. 53**	97	96. 8	86. 87	**87. 22**	**91. 23**	**87. 01**	72. 2	**89. 77**
$BFT_{w/o}LPT$	1. 1M	91. 51	88. 7	88. 1	96. 8	68. 2	73. 88	79. 51	73. 16	81. 17	71. 12	81. 22
1epoch												
LPT	792K	94. 6	88. 8	89. 4	91. 9	49	85. 2	77. 4	86. 8	81. 9	57. 8	80. 28
BFT	1. 9M	94. 4	90	89. 1	**96. 8**	84. 4	86. 4	**79. 4**	89. 7	83. 2	65. 7	85. 91
LPT-adapter	36K	**94. 61**	**90. 02**	**90. 58**	95. 85	78. 2	85. 78	76. 02	89. 58	84. 55	63. 54	84. 87
BFT-adapter	1. 13M	94. 38	89. 52	90. 47	95. 8	**96. 2**	**86. 42**	77. 66	**89. 86**	**84. 91**	**73. 65**	**87. 89**
$BFT_{w/o}LPT$	1. 1M	90. 48	85. 27	88. 55	94. 8	57. 2	71. 3	78. 4	71. 48	78. 64	71. 48	78. 76

Table 3: Performance of BFT and LPT in low-batch training. $BFT_{w/o}LPT$ means just use our partially unfrozen component. Bold face shows the best result for each column, and the underline score is the second best.

gence speed of LPT and BFT, we ran experiments with the GPT-2 Large model on SST-2 (Socher et al., 2013) and TREC. Although BFT does not improve the performance of GPT-2 as much as the RoBERTa model, it converges much faster than LPT. Figure 3 shows that BFT consistently outperforms LPT for GPT-2, and its training loss drops much faster, about two epoch ahead in most cases. However, that pattern is not so clear on other datasets.

We also tested the convergence speed on the RoBERTa model. During the training process, the epoch parameter is usually a fixed value, and the learning rate will change accordingly . For practical reasons, we tested the performance of 1 epoch and 5 epochs on the RoBERTa model. From Table 3, our BFT architecture model performs well in lower batches. Compared with other models that add information modules, using only some unfrozen modules performs well when there is only 1 epoch, but as the training time increases, it loses any advantage. Due to the excellent architecture of the adapter, both the LPT and BFT versions of the information component using adapter are superior to the information component using prompt. A model with only adapter information component (only 36K parameters) is not enough to support the model to store enough task information by comparing LPT-adapter and BFT-adapter. Each module has a task that it is good at from beginning to end. For example, the prompt information module is always better than the adapter information module on rte.

6 Conclusion

Traditional PEFT methods place their research emphasis on reducing number of adjustable parameters, but a small number of parameters does not translate directly into savings in training cost. On the other hand, LPT considers the cost of backpropagation and inserts soft prompts in the middle of a Transformer network, resulting in larger speedup as well as a small memory footprint. Our proposed model combined the idea of hard prompt and information component consisting of soft prompts or adapter, as well as a re-parameterization process, leading to both better performance and training speed. It also shows faster convergence speed than LPT. Our assumption is that parameters in the last layer of a Transformer network are the key to adapt to downstream tasks. In addition, unfreezing certain parameters show significant performance boost for complex tasks, but its additional capacity may also cause overfitting for simpler tasks. How to automatically adapt model design to downstream task information is an important research topic we plan to work on next.

7 Limitation

Our model was only tested on relatively small models (RoBERTa-Large and GPT-2 Large), which limits our ability to verify its performance on more complex natural language inference tasks. Additionally, we solely utilized prompt tuning and adapter in the addition-based methods and did not investigate the potential synergy of other addition-based methods to achieve similar effects. As a result, some of our future work will focus on verifying the performance of similar approaches. Moreover, we aim to investigate whether addition-based methods can run in synergy with specification-based models to attain faster training as well as improved performance.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*.

Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. 2021. Ppt: Pre-trained prompt tuning for few-shot learning. *arXiv preprint arXiv:2109.04332*.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 2021. Warp: Word-level adversarial reprogramming. *arXiv preprint arXiv:2101.00121*.

X Han, W Zhao, N Ding, Z Liu, and M Sun. Ptr: Prompt tuning with rules for text classification. arxiv 2021. *arXiv preprint arXiv:2105.11259*.

Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18.

Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021a. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021b. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*.

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 *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2020. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the association for computational linguistics*, 8:64–77.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035.

Jaejun Lee, Raphael Tang, and Jimmy Lin. 2019. What would elsa do? freezing layers during transformer fine-tuning. *arXiv preprint arXiv:1911.03090*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022a. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965.

Xiangyang Liu, Tianxiang Sun, Xuanjing Huang, and Xipeng Qiu. 2022b. Late prompt tuning: A late prompt could be better than many prompts. *arXiv preprint arXiv:2210.11292*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *arXiv preprint cs/0506075*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Luis Miguel Rios and Nikolaos V Sahinidis. 2013. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56:1247–1293.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2020. Adapterdrop: On the efficiency of adapters in transformers. *arXiv preprint arXiv:2010.11918*.

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 *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. 2022. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning*, pages 20841–20855. PMLR.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Ellen M Voorhees and Dawn M Tice. 2000. Building a question answering test collection. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 200–207.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Yulin Wang, Zanlin Ni, Shiji Song, Le Yang, and Gao Huang. 2021. Revisiting locally supervised learning: an alternative to end-to-end training. *arXiv preprint arXiv:2101.10832*.

Janyce Wiebe, Theresa Wilson, and Claire Cardie. 2005. Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 39:165–210.

Zhuofeng Wu, Sinong Wang, Jiatao Gu, Rui Hou, Yuxiao Dong, VG Vydiswaran, and Hao Ma. 2022. Idpg: An instance-dependent prompt generation method. *arXiv preprint arXiv:2204.04497*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Hongyu Zhao, Hao Tan, and Hongyuan Mei. 2022. Tiny-attention adapter: Contexts are more important than the number of parameters. *arXiv preprint arXiv:2211.01979*.

Zexuan Zhong, Dan Friedman, and Danqi Chen. 2021. Factual probing is [mask]: Learning vs. learning to recall. *arXiv preprint arXiv:2104.05240*.

A Appendix

A.1 Datasets

The specific information of the dataset used in our experiment is shown in the Table 4 below. Single-sentence tasks:

A.2 Implementation Details

The classification of multi-sentence tasks is complicated, so we tested 5 hand-designed hard tips and selected the best construction in Table 5.

SST-2 (The Stanford Sentiment Treebank) records the evaluation of the film and its emotional emotions. MPQA (The Multi-Perspective Question Answering) extracts from news articles from various sources. It is a sentiment classification task. Subj (Subjectivity dataset) extracts sentence from movie review document. It determines whether the sentence is subjective or objective. TREC (The Text REtrieval Conference) is a

Category	Datasets	Train	Dev	Test	Y	Type	Labels
Single-sentence	SST-2	67349	872	1821	2	sentiment	positive, negative
	MPQA	7606	1000	2000	2	opinion polarity	positive, negative
	MR	7662	1000	2000	2	sentiment	positive, negative
	Subj	7000	1000	2000	2	subjectivity	subjective, objective
	TREC	4952	500	500	6	question cls.	abbr. , entity, description, human, loc. , num.
Sentence-pair	MNLI	392702	19647	19643	3	NLI	entailment, netural, contradiction
	MRPC	3668	408	1725	2	paraphrase	equivalent, not equivalent
	QNLI	104743	5463	5463	2	NLI	entailment, not entailment
	QQP	363846	40430	390965	2	paraphrase	equivalent, not equivalent
	RTE	2490	277	3000	2	NLI	entailment, not entailment

Table 4: Specific information for each dataset

TASK	Template	label words
MRPC	Two sentences are [mask]. <S1>, <S2>.	equivalent: Yes, not equivalent: No
QNLI	Two sentences are [mask]. <S1>, <S2>.	equivalent: Yes, not equivalent: No
QQP	<S1>[mask] , <S2>	equivalent: Yes, not equivalent: No
RTE	<S1>[mask] , <S2>	entailment: Yes, not entailment: No
MNLI	<S1>[mask] , <S2>	entailment: Yes, netural: Maybe, contradiction: No

Table 5: Hard-prompt design for multi-sentence classification tasks.

question classification task, and it has 6 labels, 47 level-2 labels. Sentence-pair tasks: MNLI (The Multi-Genre Natural Language Inference Corpus) gives premises and hypotheses, judge the relationship between premises and hypotheses. According to the data sources of the training set and the test set, it is uniformly divided into two versions: matched and mismatched. Our results used the average of both as an evaluation criterion. MRPC (The Microsoft Research Paraphrase Corpus) extracts sentence pairs from newsfeeds and marks whether they were semantically equivalent. The sample distribution is uneven, with positive samples containing 68% of the total. Our results used the accuracy and F1 averages. QNLI (Qusetion-answering NLI) is a question-and-answer dataset that determines whether a question and answer are entailment. QQP (The Quora Question Pairs) extracts question pairs from the Q&A website and records whether the question pairs are semantically equivalent to each other. The distribution of QQP samples was also uneven, with positive cases accounting for 37% of the total. Our results used the accuracy and F1 averages. RTE (The Recognizing Textual Entailment datasets) is built from News and Wikipedia, and mainly determines whether the sentence pair is implied relationship.

The hyperparameters required for training are listed in Table 6. Since the adapter is directly input from the hidden layer, it has no hyperparameter num_prompt_token, and its downward projection matrix is the hidden states dimension $m \times proj\_down\_size$.

Hyperparameter	Prompt Value	Adapter Value
max_seq_length	256	256
learning_rate	1. 00E-03	1. 00E-03
weight_decay	0. 1	0. 1
logging_steps	10000	10000
num_prompt_tokens	5	
proj_down_size	128	16
warmup_rate	0. 06	0. 06
batch_size	64	64
add_prompt_layer	12	18

Table 6: Experiment hyperparameter settings

A.3 Additional experiments

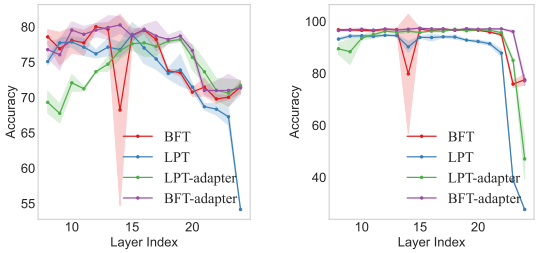
A.4 Information Component position

Since most methods only allow additional information modules to be inserted into one hidden layer of the model, we need to find a place where performance and cost are balanced. We tested the impact of insertion layer index on the performance of the LPT, BFT, LPT-adapter, and BFT-adapter models on RTE and TREC, respectively, and the results are shown in Figure 4. From the result, we chose the location that has good performance on both datasets while also close to the output. Layer 12 is selected for the prompt method and 18 is chosen for the adapter information module.

As seen in Figure 4, BFT performs slightly better than LPT regardless of the type of information

method/ behind	Tunable Parameters	rte		mrpc				SST-2	
		acc	loss	acc	F1	acc&F1	loss	acc	loss
ADD&NORM1	**10.291M**	**77.98**	**0.6164**	**79.9**	**86.56**	**83.23**	**0.6046**	95.3	**0.2217**
FFD	**10.289M**	**77.98**	0.6761	78.19	85.81	82	0.6072	**95.41**	0.224
ADD&NORM2	**1.897M**	73.65	0.7373	75.98	84.64	80.31	0.6523	94.27	0.2536
LLM_head	**1.895M**	77.26	0.7151	76.72	84.99	80.85	0.6673	94.84	0.2522

Table 7: Fine-tune schema choices. All results are after training of 5 epochs with the same random seed. The best result in each column is in bold face, and the second best is underlined. LLM_head is our base model. It only adds gradients to the Language Modeling Head. ADD&NORM2 opens gradients from layer normalization module close to LLM_head to output; FFD turn on gradients from feed-forward module to output; ADD&NORM1 opens gradients from layer normalization module to output.The bold face in the "*method behind*" column represents the BFT and its variants that we use.



(a) rte (b) trec

Figure 4: Mean and standard deviation of accuracy over 3 different random seeds, when information component is inserted at different layers in a RoBERTa-Large model.

component, especially when the information component is placed at either end of the model. This shows that our partially unfrozen component is very helpful for coordinating the training of information component together. In addition, it can be found that the optimal position of the adapter is closer to the model output than the prompt, which helps to reduce its training cost. Despite its advantage, BFT has a turning point, with performance rapidly deteriorating after that. Although an insertion point closer to output is good for reducing training cost, keeping good performance is still the main concern.

A.5 partially unfrozen component

For training cost considerations, we only allow unfrozen parameters in the last layer of the language models. To get a clear view how the backpropagation process impacts performance, we unfreeze the sublayers one by one. For example, in the RoBERTa-Large model, there are several modules of in 24th encoder layer. To facilitate comparison, we turned on gradients for all parameters from the selected module to the final output. Only the gradients after the attention module are turned on, and the experimental results are shown in Table 7. Layout of these modules are displayed in Figure 2. Based on performance, we finally selected three variants of the BFT model as shown in Table 2.