SharPT: Shared Latent Space Prompt Tuning

Bo Pang  Semih Yavuz  Caiming Xiong  Yingbo Zhou
Salesforce Research
{b.pang, syavuz, cxiong, yingbo.zhou}@salesforce.com

Abstract

Prompt tuning is an efficient method for adapting large language models, and Soft Prompt Transfer (SPoT) further narrows the gap between prompt tuning and full model tuning by transferring prompts learned from source tasks to target tasks. It is nevertheless difficult and expensive to identify the source task that provides optimal prompts. In this work, we propose to learn a shared latent space which captures a set of basis skills from a mixture of source tasks. Given an instance, its embedding queries the latent space, yielding a basis skill vector. This vector generates soft prompts, via a lightweight prompt generator, which modulates a frozen model. The latent space and prompt transformation are learned end-to-end by training on source tasks. Transfer learning from source tasks to a target task simply amounts to fine-tuning the prompt generator, accounting for roughly 0.3% parameters of the frozen backbone model, while the shared latent space is also frozen in fine-tuning. Our approach outperforms prior soft prompt methods by a significant margin on a variety of tasks such as NLI, sentence completion, QA, conference resolution, word sense disambiguation. We also find, on various model scales, our method achieves competitive performance compared to fine-tuning the full model.

1 Introduction

Adapting pre-trained large language models (LLMs) has advanced the progress in many NLP areas (Devlin et al., 2019; Raffel et al., 2020). This is typically done by finetuning all parameters of a model on a downstream task (i.e., MODEL_TUNING). This approach is however expensive, especially given the growing sizes of SOTA LLMs.

This limitation motivates recent research on parameter-efficient methods which only tune a small amount of parameters (Houlsby et al., 2019; Brown et al., 2020; Karimi Mahabadi et al., 2021; Lester et al., 2021; Li and Liang, 2021; Hambardzumyan et al., 2021). Among them, a line of research focus on the methods that modulate a frozen LLM via prompts (Liu et al., 2021). Brown et al. (2020) showed that prepending an input text with a prompt, which typically consists of a task description and/or several examples, can effectively adapt a frozen GPT-3. This approach nevertheless underperforms MODEL_TUNING and is sensitive to the choice of prompt wordings. Instead of actual text (or hard prompt), Lester et al. (2021) proposed PROMPT_TUNING, which prepends a soft prompt, consisting of \( k \) tunable tokens, to input text. The soft prompt can be optimized with gradient-based methods. PROMPT_TUNING achieves competitive performance to MODEL_TUNING when the model size is large (e.g., over 10B parameters) but still underperforms with smaller models.

SPoT (Vu et al., 2022) improves over PROMPT-
TUNING by leveraging knowledge from source tasks. They first learn a task-specific soft prompt for each task in a set of source tasks. Given a target task, they search over the set of source prompts and use the best one or some weighted combination to initialize the prompt for the target task and then tune the prompt. It further narrows the performance gap to MODEL TUNING on smaller models. But it is complicated and expensive to identify the source task that provides optimal prompts.

In this work, we propose a novel prompt-based transfer learning method, SHARPT (Shared Latent Space Prompt Tuning). Figure 1 illustrates the general idea. SHARPT assumes a shared (discrete) latent space by all source and target tasks. We call each vector in the latent space as a skill vector, since we assume each one captures a basis NLP capacity or skill after training on the source tasks. Given an instance (from either a source task or a target task), an instance encoder embeds it into an instance vector, which is then used to query the latent space to find the nearest neighbor, yielding a skill vector for this instance. A lightweight prompt generator then generates soft prompts as a function of the selected skill vector. The soft prompts condition a frozen LLM. The latent space and prompt generator are learned end-to-end on a mixture of source tasks. In target task training, the latent space is frozen and only the prompt generator is tuned.

SHARPT retains the key advantage of prior prompt methods, parameter-efficiency. It only updates approximately 0.1% to 0.3% parameters compared to MODEL TUNING. Different from prior methods, we add an instance encoder to encode each instance. The instance encoder is lightweight and frozen in all scenarios.

SHARPT and SPoT both exploit a generic idea, leveraging knowledge shared across tasks. The approaches to achieve this are however distinctly different. SPoT assumes task-to-task transfer based on task-level prompts and the knowledge is encoded in task prompts. It is not straightforward to identify a source prompt for a target task. They illustrated two approaches: (1) SPoT-Oracle and (2) SPoT-Retrieval. SPoT-Oracle involves using oracle test labels and expensive search (e.g., 48 times more expensive than regular prompt tuning in their experiments). In SPoT-Retrieval, they first tuned a task prompt for each source and target task independently and retrieved a prompt based on prompt similarity. Note that the retrieval tuning is only for searching a source prompt, which is in addition to final prompt tuning on the target task. In contrast, SHARPT assumes the knowledge is encoded in a shared latent space and utilizes instance-level prompts, which are generated based on latent vectors from the shared space. These designs make source-to-target transfer simple. We learn the shared latent space with all source tasks in a single training run. Also, the tuning on the target task only requires a single run. Given an instance from a target task, we use the instance embedding to identify a skill vector, learned from all source tasks, which is then transformed to soft prompts.

In summary, we design an instance-prompt-based method by learning a shared skill latent space. We apply SHARPT to a diverse set of tasks covering diverse domains and task categories. We find that our method outperforms prior prompt-based methods and matches full-model-tuning across model scales.

2 Method

Suppose we have a task with data $T = \{(x, y)\}$ and a pre-trained LLM $P_0$. MODEL TUNING updates $\theta$ to minimize $L(\theta) = -\log P_0(y|x)$.

Prompt Tuning prepends to $x$ a soft prompt, $p \in \mathbb{R}^{L \times d}$, which has $L$ vectors of size $d$. It then optimizes $p$ by minimizing $L(p) = -\log P_0(y|p, x)$.

SHARPT assumes there exists a discrete latent space, consisting of a set of skill vectors $E = \{e_i \in \mathbb{R}^m\}_{i=1}^K$ with $K$ vectors in total. The soft prompt is a simple transformation of one of the skill vectors $e_i$, that is, $p = f_{\alpha}(e_i)$. The transformation or prompt generator ($f_{\alpha}$) is a light-weight MLP.

$$e_i = \text{Tanh}(W_1e_i + b_1), \quad p_l = W_2(z_l + e_i) + b_2$$

(1)

where $z_l \in \mathbb{R}^d$ is the position embedding for the $l$th token (and randomly initialized in training) and $W_1 \in \mathbb{R}^{d \times m}, W_2 \in \mathbb{R}^{d \times d}$. Then we have the soft prompt $p = \{p_l\}_{l=1}^L$.

Given $x$, we infer its skill vector by (1) embedding it via a frozen instance encoder (e.g., SimCSE BERT-base), which yields $e_x^{(0)}$; (2) querying $E$ to find the nearest neighbor. Formally, that is,

$$e_x^{(1)} = e_k, \quad k = \arg \min_{i \in [K]} \|e_x^{(0)} - e_i\|_2.$$  

(2)

For a target task, our method is then trained with the following loss,

$$L(\alpha) = -\log P_0(y|f_{\alpha}(e_k), x).$$  

(3)

$^1$Summation over the data is omitted for notation clarity.
In target task training aforementioned, \( E \) is known and fixed. We next specify how to learn it from source tasks. Suppose we have \( N \) source tasks, \( \{T_j^{(s)}\}_{j=1}^N \). We simply mix all tasks together, \( T^{(s)} = \bigcup_{j=1}^N T_j^{(s)} \). Given \( x \in T^{(s)} \) and its embedding \( e_x^{(0)} \), \( E \) is learned with the following loss, 

\[
\mathcal{L}(E) = \left\|\text{sg}(e_x^{(0)}) - e_k\right\|_2^2, \tag{4}
\]

where \( \text{sg}(\cdot) \) is a stop gradient operator and \( e_k \) is defined in Equation (2). The overall loss in source task learning is, 

\[
\mathcal{L}(\alpha, E) = \mathcal{L}(\alpha) + \mathcal{L}(E) \tag{5}
\]

In summary, the forward pass for training on source and target tasks are exactly the same (also see Figure 1). The only difference is the loss function, Equation 5 (source) versus Equation 3 (target).

3 Experiments

High-to-Low Resource Transfer In this setting, the target tasks are low-resource tasks (less than 10K training examples), while the source tasks are high-resource tasks. It consists of 25 tasks in total. There are 15 source tasks (e.g., DocNLI, DROP) and 10 target tasks (e.g., BoolQ, CoLA). Please see Appendix A for the complete list or Table 1 for the target tasks. We keep the setting to be almost the same as a major experiment in Vu et al. (2022) for a fair comparison, with the exception that we exclude C4 from the source task since it is a much larger dataset than other tasks. Excluding C4 does not affect SPoT performance since it does not provide an optimal source prompt for any target task.

Transfer across Different Task Categories We here investigate the transferability from datasets in some task categories to datasets in other held-out task categories. Following Sanh et al. (2022), we assume datasets in each category measures a general NLP ability, and use the same taxonomy defined in Sanh et al. (2022). The source tasks include (1) QA tasks: ReCoRD, SQuAD, DROP, MultiRC, and RACE; (2) sentiment analysis tasks: Yelp-2 and SST-2; (3) a paraphrase detection task: QQP; (4) a semantic similarity task: CXC. The target tasks include (1) a sentence completion task: COPA; (2) NLI tasks: CB and RTE; (3) a coreference resolution tasks: WSC; (4) a word sense disambiguation task: WiC.

Training Details As in prior works (Raffel et al., 2020; Lester et al., 2021), all datasets are converted to a text-to-text format. All experiments are conducted with T5-base-LM-adapted as the backbone unless stated otherwise. We use a SimCSE (Gao et al., 2021) model (BERT-base) as the instance encoder. Since the instance encoder is always frozen, we can pre-compute the embeddings of all instances and only keep the embeddings. However, we find that memory and time saved in this approach is negligible. In source task training, the model (skill latent space and prompt generator) is simply tuned on the mixture of all source tasks for each setting. The model is tuned for 80K steps. In learning and testing on target tasks, we closely follow the procedure in Vu et al. (2022). The model is tuned for 100K on each target task. We save a checkpoint every 500 steps and report results on the checkpoint with the highest validation performance. The prompt generator generates 64 soft tokens. The following hyperparameters are shared in all target and source task training: learning rate (0.3), the number of warmup steps (4000), optimizer (Adam).

4 Results

High-to-Low Resource Transfer The results are shown in Table 1. We first compare our method, SHARPT, to methods with comparable compute- and parameter-efficiency, PROMPTTUNING and SPoT-Retrieval. Our method has a clear improvement over the two methods across most tasks and on the average performance. We next compare SHARPT with much more expensive methods, SPoT-Oracle and MODELTUNING. Note that SPoT-Oracle is significantly more expensive than our method since it tunes on each target task with each possible task prompt (e.g., it requires roughly 48 times more training time), and utilizes oracle labels. While being much more efficient, SHARPT matches or outperforms SPoT-Oracle. Also, our method performance is on par with the MODELTUNING performance which requires to tune the entire model. These results indicate SHARPT is an efficient and competitive approach.

Transfer across Different Task Categories The results are shown in Table 2. Our method outperforms both PROMPTTUNING and SPoT methods. For instance, removing the instance encoder in training (by pre-computing the instance embeddings) does not allow a larger batch size compared to including the instance encoder.
Table 1: Results on the high-to-low transfer learning setting. Methods in the upper panel are significantly more expensive than those in the lower panel. The best performance is in **bold**, and the second best is underlined.

<table>
<thead>
<tr>
<th>Model</th>
<th>BoolQ</th>
<th>CB</th>
<th>CoLA</th>
<th>COPA</th>
<th>CR</th>
<th>MRPC</th>
<th>RTE</th>
<th>STS-B</th>
<th>WiC</th>
<th>WSC</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelTuning</td>
<td><strong>81.4</strong></td>
<td>94.0</td>
<td>51.1</td>
<td><strong>71.2</strong></td>
<td>94.1</td>
<td>87.5</td>
<td><strong>81.5</strong></td>
<td>89.4</td>
<td>68.3</td>
<td>80.8</td>
<td><strong>79.9</strong></td>
</tr>
<tr>
<td>SPoT-Oracle</td>
<td>77.6</td>
<td><strong>97.0</strong></td>
<td>55.6</td>
<td>69.3</td>
<td>93.9</td>
<td>88.7</td>
<td>74.7</td>
<td><strong>90.0</strong></td>
<td>70.2</td>
<td>77.2</td>
<td>79.4</td>
</tr>
<tr>
<td>PromptTuning</td>
<td>73.0</td>
<td>92.7</td>
<td>52.9</td>
<td>56.7</td>
<td>93.5</td>
<td>86.1</td>
<td>68.7</td>
<td>88.1</td>
<td>63.6</td>
<td>71.5</td>
<td>74.7</td>
</tr>
<tr>
<td>SPoT-Retrieval</td>
<td>74.2</td>
<td>95.4</td>
<td>54.8</td>
<td>58.3</td>
<td>93.6</td>
<td>88.4</td>
<td>71.6</td>
<td><strong>90.0</strong></td>
<td>66.7</td>
<td>72.9</td>
<td>76.6</td>
</tr>
<tr>
<td>SHARPT</td>
<td>78.9</td>
<td>94.6</td>
<td><strong>58.2</strong></td>
<td>67.0</td>
<td><strong>94.5</strong></td>
<td>89.7</td>
<td>79.4</td>
<td>89.1</td>
<td>68.8</td>
<td>81.6</td>
<td><strong>80.2</strong></td>
</tr>
</tbody>
</table>

Table 2: Results on transferring across task categories.

<table>
<thead>
<tr>
<th>Model</th>
<th>COPA</th>
<th>CB</th>
<th>RTE</th>
<th>WSC</th>
<th>WiC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelTuning</td>
<td><strong>71.2</strong></td>
<td>94.0</td>
<td><strong>81.5</strong></td>
<td><strong>80.8</strong></td>
<td>68.3</td>
</tr>
<tr>
<td>SPoT-Oracle</td>
<td>63.0</td>
<td>92.9</td>
<td>72.0</td>
<td>77.2</td>
<td><strong>70.2</strong></td>
</tr>
<tr>
<td>PromptTuning</td>
<td>56.7</td>
<td>92.7</td>
<td>68.7</td>
<td>71.5</td>
<td>63.6</td>
</tr>
<tr>
<td>SPoT-Retrieval</td>
<td>61.2</td>
<td>89.4</td>
<td>71.4</td>
<td>73.6</td>
<td>66.7</td>
</tr>
<tr>
<td>SHARPT</td>
<td>65.0</td>
<td><strong>94.6</strong></td>
<td>79.4</td>
<td>79.0</td>
<td>69.8</td>
</tr>
</tbody>
</table>

Figure 2: Results on models of different sizes.

The improvement over SPoT methods is larger in this setting than in the high-to-low transfer setting. This might be because SPoT relies more on knowledge shared by tasks in the same category, while SHARPT learns a *shared latent space across all source tasks* and is more suitable to leverage knowledge shared across datasets of different categories.

**Across Model Scales** In the experiments above, we show that our method can close the performance gap between full model tuning and prompt-based methods on a mid-sized model, T5-base (220M). Here conducts experiments with larger models, T5-large (800M) and T5-xl (3B), and compare SHARPT to ModelTuning and PromptTuning. As shown in Figure 2, SHARPT matches or slightly outperforms ModelTuning under the three model scales. Our method also shows considerable improvements over PromptTuning.

**Ablations** We ablate two key components of SHARPT: (1) training on source tasks; (2) skill latent space that captures shared knowledge. See the results in Table 3. Clearly, knowledge learned from source tasks and encoded in the latent space is critical for target task performance.

<table>
<thead>
<tr>
<th>Model</th>
<th>BoolQ</th>
<th>CB</th>
<th>CoLA</th>
<th>COPA</th>
<th>CR</th>
<th>MRPC</th>
<th>RTE</th>
<th>STS-B</th>
<th>WiC</th>
<th>WSC</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Source Task Training</td>
<td>64.3</td>
<td>89.3</td>
<td>10.3</td>
<td><strong>58.0</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Latent Space</td>
<td>67.9</td>
<td>82.4</td>
<td>17.6</td>
<td>61.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Ablation results.

Figure 3: A heatmap of task relations based on skill vector usage of each task.

**Task Relations** We investigate if the latent space captures source and target task relations to allow knowledge transfer. Each instance queries the latent space and selects one latent skill. We convert this selection to a one-hot vector and treat it as an instance encoding. A task representation is the average of instance encodings in the task. The cosine similarity between two task representations is computed as their relation. The relations between source and target tasks are visualized in Figure 3. It seems that more complicated source tasks such as QA and NLI tasks transfer more knowledge to target tasks via the skill latent space.

5 Conclusion

We introduce SHARPT, which learns a shared latent space which captures a set of basis NLP capacities from a mixture of source tasks. Target instance queries this space to retrieve a skill vector, which then generates prompt tokens to condition a frozen LLM. Our approach outperforms prior soft prompt methods by a significant margin on a variety of tasks. Our method also matches full-model-tuning across model scales.
Limitations

Although our method is much simpler than SPOT, PROMPT TUNING is still arguably the simplest method for adapting LLMs to downstream tasks. It would be a fruitful research direction to design transfer learning approaches that retain (or even improve) our method’s performance and meanwhile further simplify our method, getting closer to the simplicity of PROMPT TUNING.

References


A Source and Target Tasks in the High-to-Low Resource Transfer Setting

The source tasks include DocNLI (Yin et al., 2021), Yelp-2 (Wang et al., 2018), MNLI (Williams et al., 2018), QQP (Iyer et al., 2017), QNLI (Wang et al., 2018), ReCoRD (Zhang et al., 2018), CXC (Parekh et al., 2021), SQuAD (Rajpurkar et al., 2016), DROP (Dua et al., 2019), SST-2 (Socher et al., 2013), WinoGrande (Sakaguchi et al., 2021), HellaSWAG (Zellers et al., 2019), MultiRC (Khashabi et al., 2018), CosmosQA (Huang et al., 2019), RACE (Lai et al., 2017).

The target tasks includeBoolQ (Clark et al., 2019), CB (De Marneffe et al., 2019), CoLA (Warstadt et al., 2019), COPA (Roemmele et al., 2011), CR (De Marneffe et al., 2019), MRPC (Dolan and Brockett, 2005), RTE (Dagan et al., 2005), STS-B (Cer et al., 2017), WiC (Pilehvar and Camacho-Collados, 2019), WSC (Levesque et al., 2012).