Prompt turning report

Prefix-Tuning: Optimizing Continuous Prompts for Generation

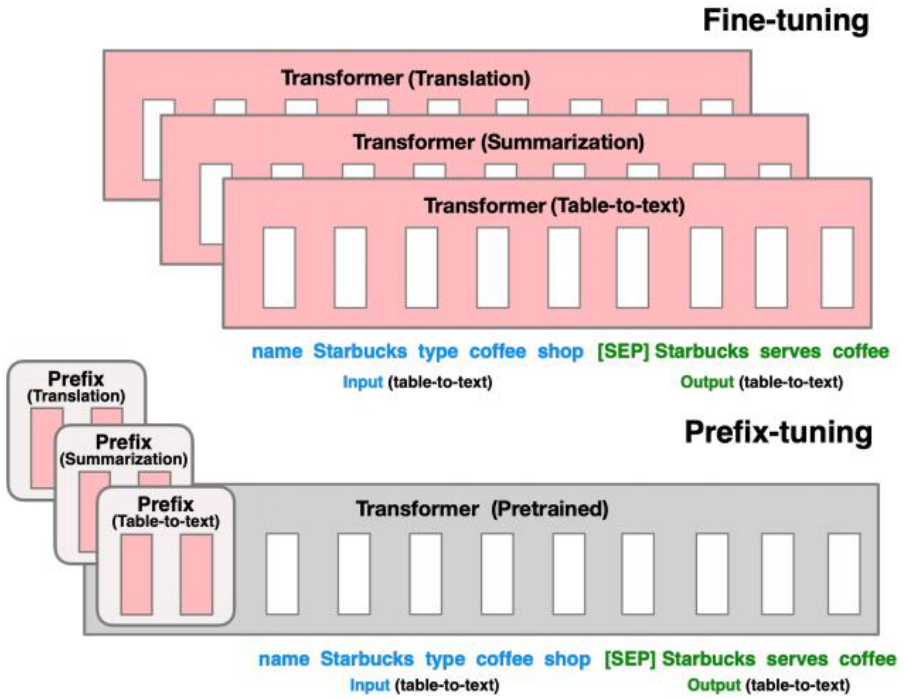
论文链接：https://arxiv.org/pdf/2101.00190.pdf

代码链接：https://github.com/XiangLi1999/PrefixTuning

Background and Motivation

大型预训练模型的训练成本非常高

1. Fine-tuning：是一种常用的策略，用于调整大型语言模型以适应特定的下游任务，但这种方法通常涉及到对模型的所有参数进行修改，这意味着对于每个不同的任务，都需要保存一个调整后的模型副本，贵且浪费
2. Lightweight fine-tuning：固定住绝大部分的预训练参数，修改预训练模型的小部分模块。该方法最困难的地方是**识别模块中高表现的部分**以其需要调节的预训练参数。（adapter-tuning，预训练语言模型的各层之间插入额外的任务特定层）



从fine-turning角度出发的，本文提出了**prefix-tunning**,每个任务添加少量的任务特定的参数，并将这些**prefix**直接加到模型**输入的开始处**，模型输入序列的一部分。（模块化的，可以避免数据交叉污染）

它不需要对大型Transformer模型的所有参数进行微调，对于每个新任务只需**添加少量参数**。与传统的微调方法相比，prefix-tuning显著减少了每个任务所需的**额外存储空间**。

Problem Statement

主要考虑table-to-text任务和summarization任务（conditional generation），**输入context x，输出sequence of tokens y**，基于Transformer架构的模型，用于预测条件生成任务中下一个token的概率分布

$z = [x; y]$ 是最终的激活序列，X_{idx} 和 Y_{idx} 表示原序列的indices。$h_i \in \mathbb{R}^d$ 表示在时间步 i 下的激活值。

$$h_i = LM_\phi(z_i, h_{<i})$$

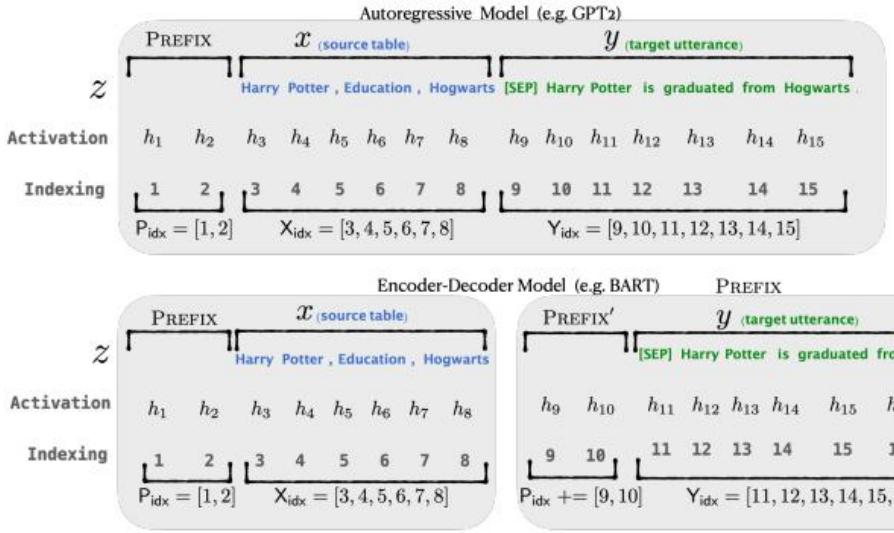
h_i 的最后一层用来计算下一个token的分布：

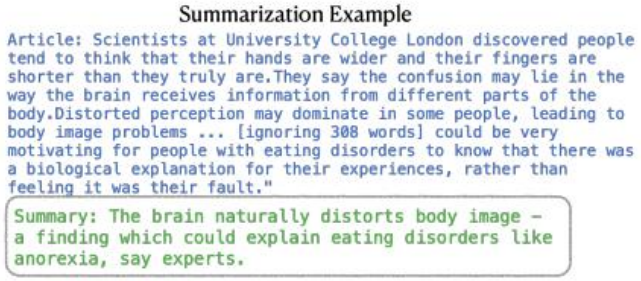
$$p_\phi(z_{i+1}|h_{<i}) = \text{softmax}(W_\phi h_i^{(n)})$$

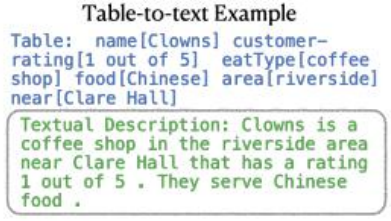
如果正常fine-tuning，公式如下：

$$\max_\phi \log p_\phi(y \mid x) = \sum_{i \in \text{Y}_{\text{idx}}} \log p_\phi\left(z_i \mid h_{<i}\right)$$

Method







添加一个prefix，decoder表示为 $z = [\text{prefix}; x; y]$，encoder-decoder模型表示为 $z = [\text{prefix}; x; \text{prefix}'; y]$。

prefix-tuning初始化一个训练的虚拟序列，记作$P_\theta \in \mathbb{R}^{|\text{Pidx}| \times \dim(h_i)}$，这部分参数叫做prefix parameters。

P_idx表示prefix indices序列；$|P_idx|$ 表示prefix的长度。

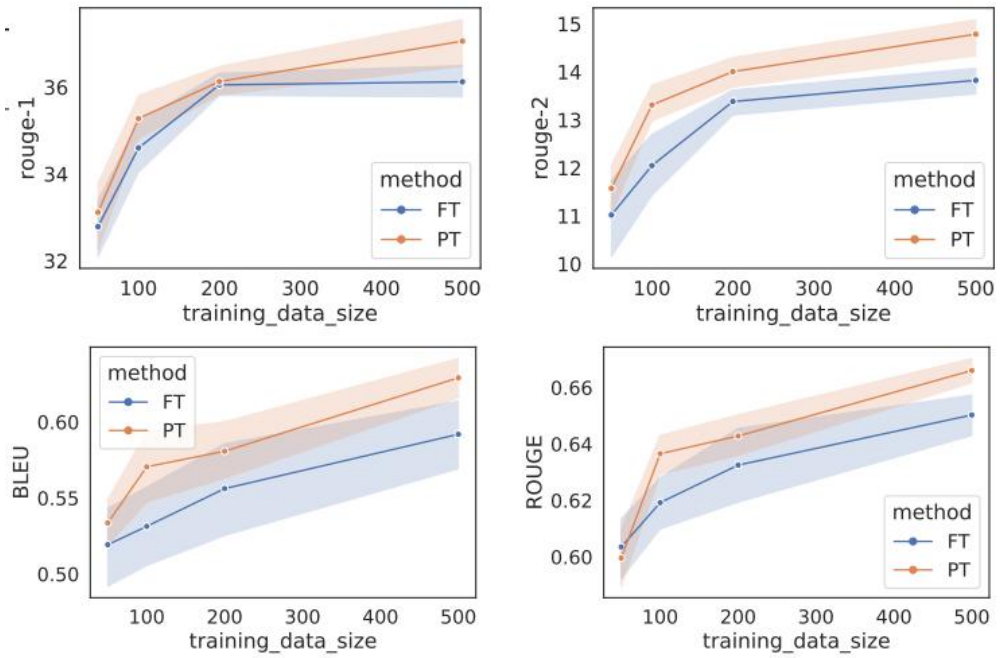
training objective在Fine-tuning阶段，但语言模型的参数 θ 固定，仅仅prefix参数 θ是可以训练的。因此 h_i是可以继承的 P_θ 的参数，当 $i \in P_idx$ 时，h_i 由 P_θ 直接决定；对于 $i \notin P_idx$，由于prefix activations始终存在left context因此可以决定 h_i。

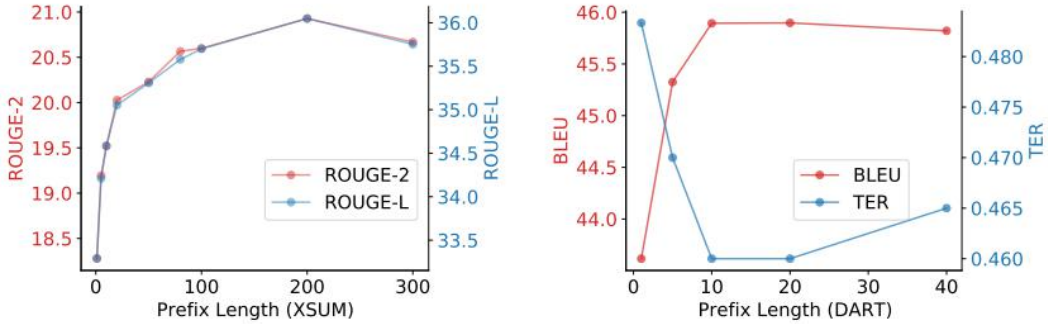
$$h_i = \begin{cases} P_\theta[i, :], & \text{if } i \in \mathsf{P}_{\mathsf{idx}}, \\ \mathrm{LM}_\phi(z_i, h_{<i}), & \text{otherwise.} \end{cases}$$

直接更新P的参数会导致优化的不稳定,因此使用MLP来reparametrize矩阵

$$P_\theta[i, :] = MLP_\theta(P'_\theta[i, :])$$

Experiments and results



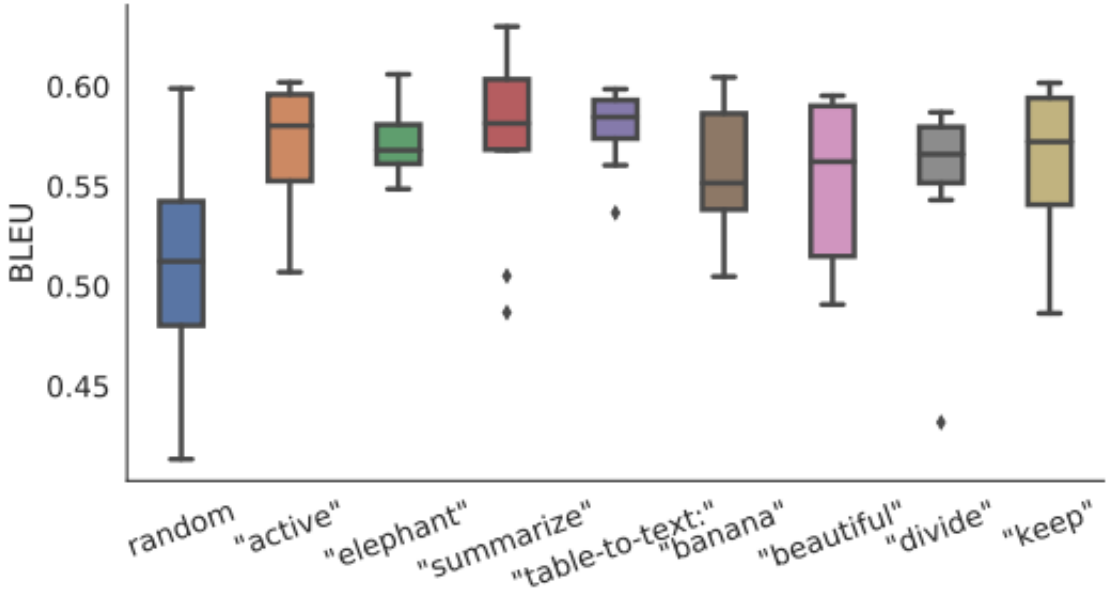


	news-to-sports			within-news		
	R-1 ↑	R-2 ↑	R-L ↑	R-1 ↑	R-2 ↑	R-L ↑
FINE-TUNE	38.15	15.51	30.26	39.20	16.35	31.15
PREFIX	39.23	16.74	31.51	39.41	16.87	31.47

Table 3: Extrapolation performance on XSUM. Prefix-tuning outperforms fine-tuning on both news-to-sports and within-news splits.

Prefix-tuning能用更少的参数达到较有竞争力的结果。在Low-data阶段（训练样本数较少）， prefix-tuning相比较fine-tuning更有优势。在Extrapolation方面，prefix-tuning也比fine-tuning的表现更好,不同任务prefix长度也不是越长越好

Experiments and results



使用summarize和table-to-text激活效果最好。

1.最小值（箱子的下边界）：这是除去异常值外的最小BLEU分数。

2.下四分位数（箱子底部的边界）：表示所有数据中有25%的数据低于这个值。

3.中位数（箱子中间的线）：数据的中间值，意味着一半的数据分布在这个值以上，另一半在这个值以下。

4.上四分位数（箱子顶部的边界）：表示所有数据中有25%的数据高于这个值。

5.最大值（箱子的上边界）：这是除去异常值外的最大BLEU分数。

GPT Understands, Too （P-Turning）

论文链接：https://arxiv.org/abs/2103.10385

代码链接：https://github.com/THUDM/P-tuning

Background and Motivation

Background

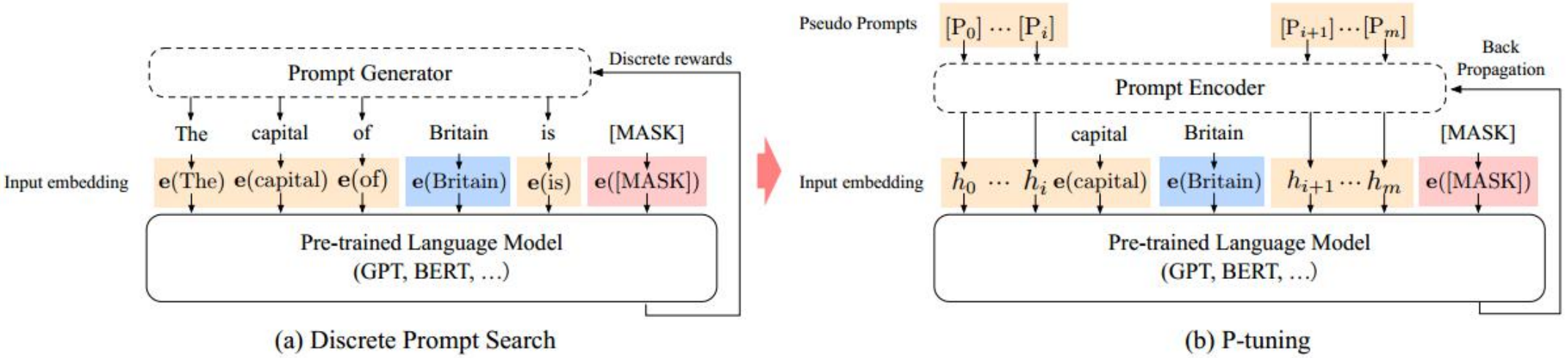
1. 通过给模型提供任务特定的prompt（**离散的**），可以进一步提升性能。然而，传统的方法经常面临稳定性问题，微小的prompt变化可能导致性能显著下降。
   1. 大模型的 Prompt 构造方式严重影响下游任务的效果
   2. 人工设计的模版的变化特别敏感，加一个词或者少一个词，或者变动位置都会造成比较大的变化。
2. 近来的自动化搜索模版工作成本也比较高， 以前这种 离散化的 token 的搜索出来的结果可能并不是最优的。

Motivation

1. 是解决离散prompt的不稳定性，通过引入**可训练的连续prompt（**P-Tuning），设计了一种连续可微的pseudo token（同Prefix-Tuning类似）。
2. P-tuning重新审视了关于template的定义，放弃了“**template由自然语言构成**”这一常规要求，从而将template的构建转化为**连续参数优化**问题

这篇文章是从prompt角度出发的，本质上来说，作者并不关心template长什么样，只需要知道template由哪些token组成，该插入到哪里，插入后能不能完成下游任务，输出的候选空间是什么。模版是不是**自然语言组成的,根本没影响**

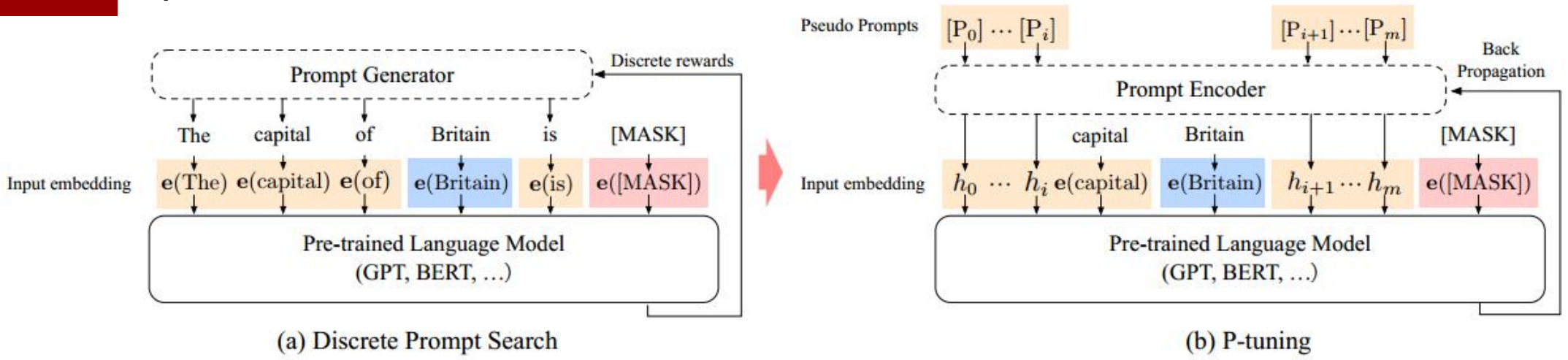
Method



橙色的是prompt，蓝色的是输入的context,红色的是预测的。例子为：“The capital of Britain is [MASK]"，这里面，”The capital of ... is ...“是prompt，”Britain“是context，以及[MASK]是目标

通过实验发现用一个prompt encoder来编码会收敛更快，效果更好。即用一个LSTM+MLP去编码这些Pseudo Prompts以后，再输入到模型。

Experiments and results



(a) Discrete Prompt Search (b) P-tuning

1. P-turning模板T被定义为：$T = [P_0 : i], x, [P_{i+1:m}], y$，每一个元素都是token，$x$是输入，$y$是预测的输出，$P$为Pseudo Prompts。
2. 这些embedding层之后，上面的模板T可以被表示为：$e([P_0 : i]), e(x), e([P_{i+1:m}]), e(y)$ 即 $h_0, ..., h_i, e(x), h_{i+1}, ..., h_m, e(y)$。
3. 在P-tuning中，不是使用的一个普通的prompt tokens，而是使用一个$[P_i]$作为一个Pseudo Prompts，其用途类似于占位符：被映射为$h_0, ..., h_i, e(x), h_{i+1}, ..., h_m, e(y)$，这里面，$h_0(0 \leq i < m)$ 是训练中的embedding tensors。这样的好处（使用一个普通的token，替换成一个连续的embedding tensors）在于，其可以促进我们更好地学习和掌握那些与“mask”。
4. 最后，基于下游的损失函数 L（根据任务的不同，损失函数的具体形式也会不一样），通过优化连续embedding的prompt $h_i(0 \leq i < m)$：

$$\hat{h}_{0:m} = \arg\min_{h} L(M(x, y))$$

Experiments and results

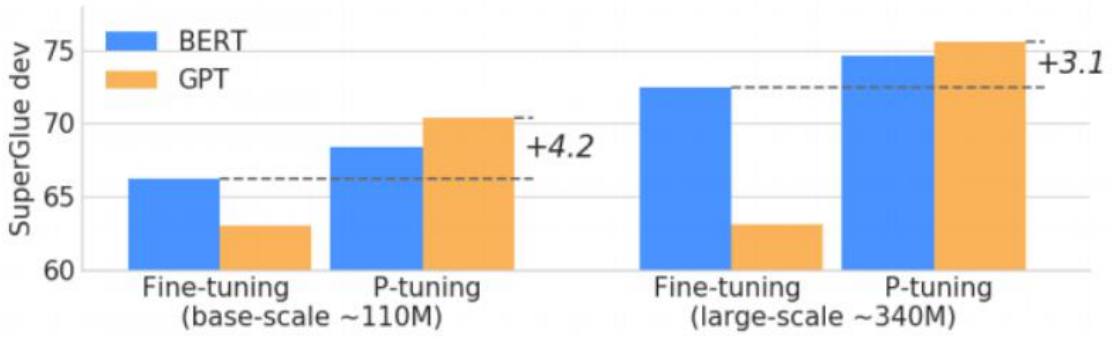
GPT Understands, Too

Prompt type	Model	P@1
Original (MP)	BERT-base	31.1
	BERT-large	32.3
	E-BERT	36.2
Discrete	LPAQA (BERT-base)	34.1
	LPAQA (BERT-large)	39.4
	AutoPrompt (BERT-base)	43.3
P-tuning	BERT-base	48.3
	BERT-large	**50.6**

Model	MP	FT	MP+FT	P-tuning
BERT-base (109M)	31.7	51.6	52.1	52.3 (+20.6)
-AutoPrompt (Shin et al., 2020)	-	-	-	45.2
BERT-large (335M)	33.5	54.0	55.0	54.6 (+21.1)
RoBERTa-base (125M)	18.4	49.2	50.0	49.3 (+30.9)
-AutoPrompt (Shin et al., 2020)	-	-	-	40.0
RoBERTa-large (355M)	22.1	52.3	52.4	53.5 (+31.4)
GPT2-medium (345M)	20.3	41.9	38.2	46.5 (+26.2)
GPT2-xl (1.5B)	22.8	44.9	46.5	54.4 (+31.6)
MegatronLM (11B)	23.1	OOM*	OOM*	**64.2** (+41.1)

* MegatronLM (11B) is too large for effective fine-tuning.

p-turning 对GPT增益极高。在相同参数规模下，bert的在nlu任务上的效果，超过gpt很多；但是在p-tuning下，gpt可以取得超越bert的效果

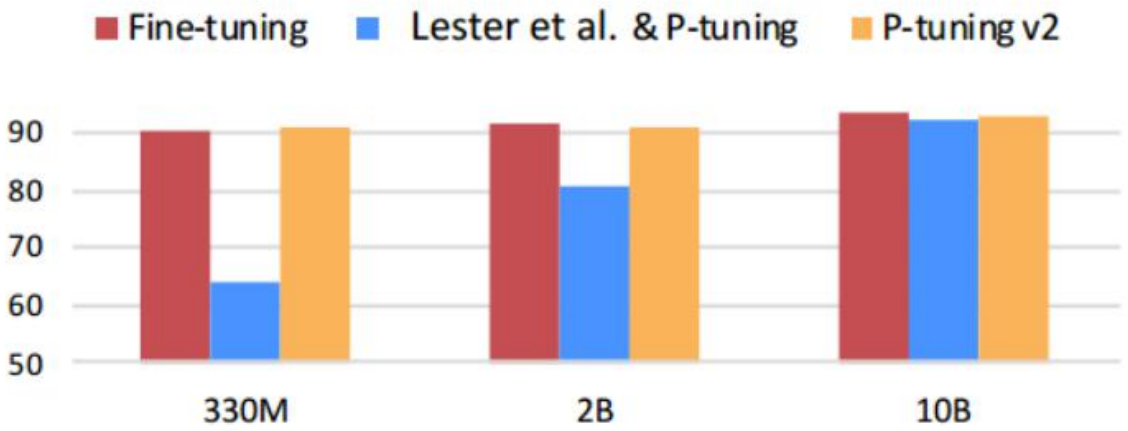


P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Across Scales and Tasks

论文链接：https://arxiv.org/pdf/2110.07602.pdf

代码链接：https://github.com/THUDM/P-tuning-v2

Background and Motivation



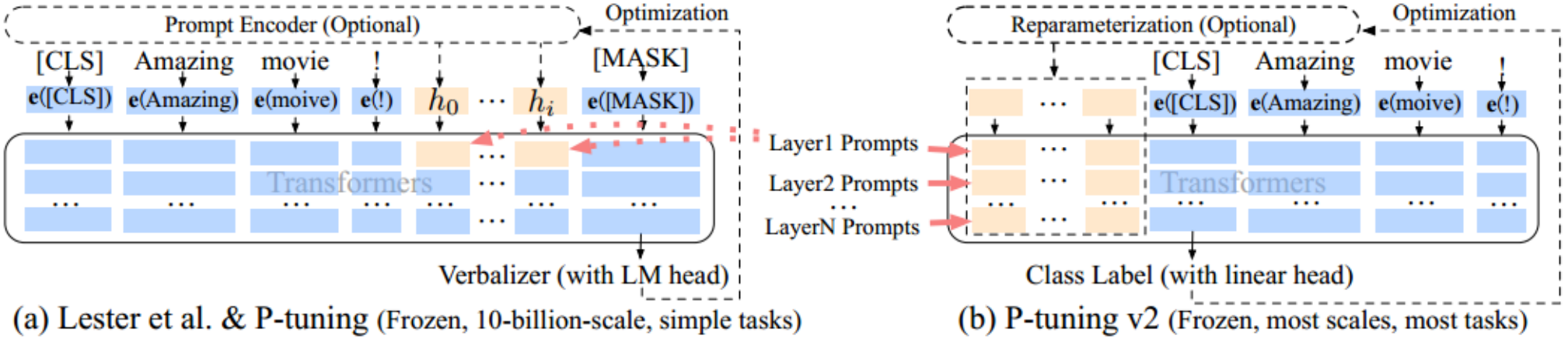
Background

1. Fine-tune LLM现在虽然可以为下游任务提供很好的表现，但是这个过程会受到算力、时间的限制，为多个task同时储存它们各自的权重也是比较贵的事情
2. 以前关于Prompt-Tuning的工作(P-Tuning v1)小的预训练模型中表现不够理想,参数比较少的模型效果不是很友好（10B以下）， 并且对hard sequence 的 labeling task 表现也不是很好。
3. 在V1 中在input的上做修改，作者认为着面临两个问题
    1. 由于序列长度的限制，可调节的参数不多
    2. 模型层数很多，随着层数的提升，这个prompt可能不会直接影响最重的结果。

Motivation

为了解决那三个问题，本文提出P-tuning-V2模型（对Prefix-tuning的扩展），发现稍微改进优化后的prompt-tuning依然可以达到与传统Fine-tuning一样甚至更好的效果；提出Deep Prompt Tuning，进一步对pseudo continuous embedding进行深度表征

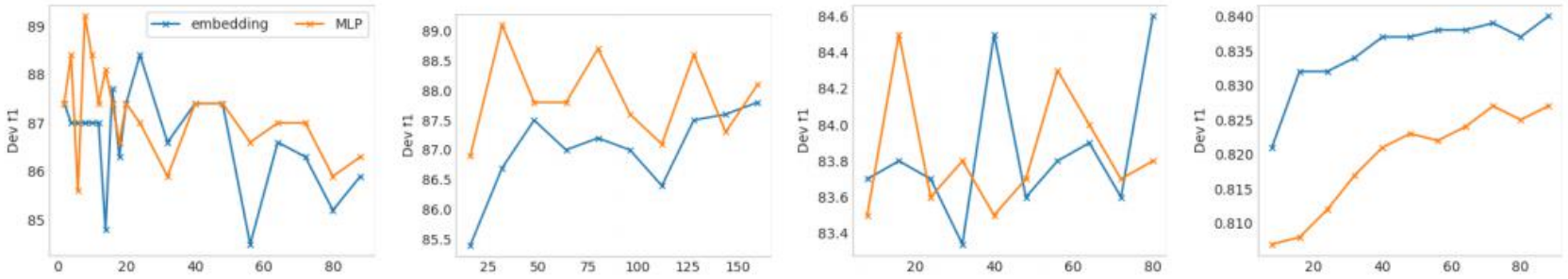
Method



P-tuning-V2不是一个新的方法，是一个优化和改良的Prefix-tuning版本。

- **取消了reparamerization**：对pseudo token，不再使用BiLSTM或MLP进行表征，直接对pseudo token对应的深层模型的参数进行微调。（对于NLU任务，文章发现这个技术的结果依赖任务和数据集）
- **Prompt Length**：作者认为数据prompt的长度与任务的复杂度有关，复杂一些的任务可能需要更多的token
  - 比如简单的分类喜欢小于20的Prompts,困难的序列标注问题喜欢较长的Prompts(大约100)
- **Multi-task Learning**：基于多任务数据集的Prompt进行预训练，然后再适配的下游任务
- **Classification Head**: 可以在[CLS]部分输出sentence-level class，以及每个token位置上输出token-level class。
  - 在tokens上应用一个随机初始化的[CLS]，类似于Bert中的处理方式。这种[CLS]直接对tokens进行分类，而不是依赖于一个verbalizer将模型的输出映射到预定的词汇上。这样的设计允许模型直接为序列标注任务产生预测，而不必依赖于额外的verbalizer步骤。

Method



(a) NLI: RTE (b) NER: CoNLL04 (c) MQA: BoolQ (d) SRL: CoNLL12

Figure 4: Ablation study on prompt length and reparamerization using RoBERTa-large. The conclusion can be very different given certain NLU task and dataset. (MQA: Multiple-choice QA)

Method	Task	Re-param.	Deep PT	Multi-task	No verb.
P-tuning (Liu et al., 2021)	KP NLU	LSTM	-	-	-
PROMPTTUNING (Lester et al., 2021)	NLU	-	-	✓	-
Prefix Tuning (Li and Liang, 2021)	NLG	MLP	✓	-	-
SOFT PROMPTS (Qin and Eisner, 2021)	KP	-	✓	-	-
P-tuning v2 (Ours)	NLU SeqTag	*(depends)*	✓	✓	✓

Some datasets (like RTE and CoNLL04) see consistent improvement with the use of MLP for reparameterization, other datasets (like BoolQ and CoNLL12) do not benefit as much, and the use of MLP could even lead to worse performance.

- 技术上区别：P-tuning-v2把重参数化去掉了。而Prefix-tuning是有mlp进行转换，再输入到模型中。
- 模型和任务不一样：Prefix-tuning主要着重在decoder only和encoder-decoder在文本生成任务中。P-tuning-v2则是不限于生成任务，是应用在所有nlp任务中（包括nlu和nlg）

Experiments and results

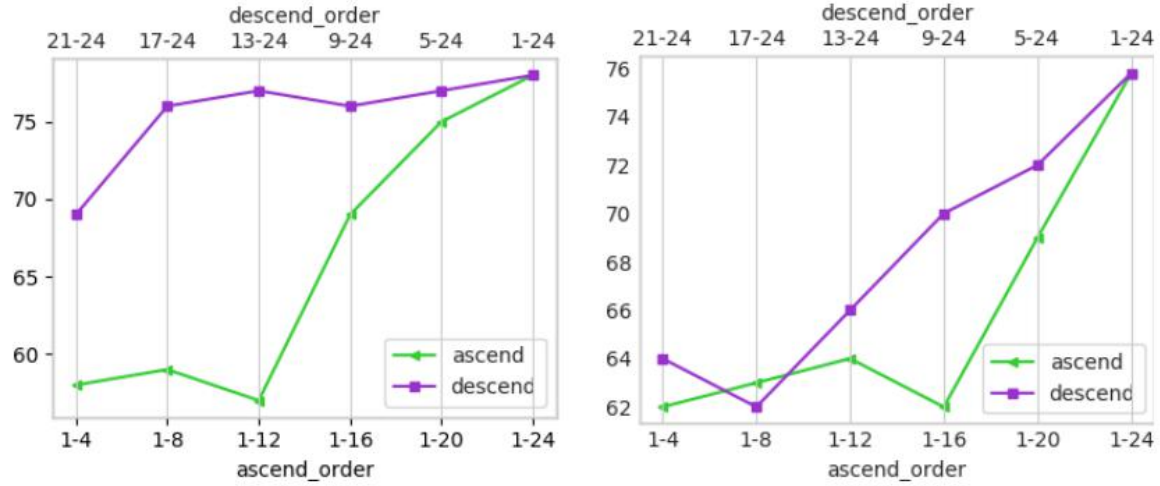
在小于10B的模型上，P-tuning v2明显超过了P-tuning，并与微调的性能相符。

	#Size	CoNLL03				OntoNotes 5.0				CoNLL04			
		FT	PT	PT-2	MPT-2	FT	PT	PT-2	MPT-2	FT	PT	PT-2	MPT-2
$\text{BERT}_{\text{large}}$	335M	**92.8**	81.9	90.2	91.0	**89.2**	74.6	86.4	86.3	85.6	73.6	84.5	**86.6** (+1.0)
$\text{RoBERTa}_{\text{large}}$	355M	92.6	86.1	92.4	**92.8** (+0.2)	**89.8**	80.8	89.4	**89.8** (+0.0)	88.8	76.2	87.3	**90.6** (+0.8)
$\text{DeBERTa}_{\text{xlarge}}$	750M	93.1	90.2	93.1 (+0.0)	**93.1** (+0.0)	90.4	85.1	90.4 (+0.0)	**90.5** (+0.1)	89.1	82.4	86.5	**90.1** (+1.0)

	#Size	SQuAD 1.1 dev								SQuAD 2.0 dev							
		FT		PT		PT-2		MPT-2		FT		PT		PT-2		MPT-2	
		EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
$\text{BERT}_{\text{large}}$	335M	**84.2**	**91.1**	1.0	8.5	77.8	86.0	82.3	89.6	**78.7**	**81.9**	50.2	50.2	69.7	73.5	72.7	75.9
$\text{RoBERTa}_{\text{large}}$	355M	**88.9**	**94.6**	1.2	12.0	88.1	94.1	88.0	94.1	**86.5**	**89.4**	50.2	50.2	82.1	85.5	83.4	86.7
$\text{DeBERTa}_{\text{xlarge}}$	750M	90.1	95.5	2.4	19.0	**90.4** (+0.3)	**95.7** (+0.2)	89.6	95.4	88.3	91.1	50.2	50.2	**88.4** (+0.1)	**91.1** (+0.0)	88.1	90.8

	#Size	CoNLL12				CoNLL05 WSJ				CoNLL05 Brown			
		FT	PT	PT-2	MPT-2	FT	PT	PT-2	MPT-2	FT	PT	PT-2	MPT-2
$\text{BERT}_{\text{large}}$	335M	84.9	64.5.	82.5	**85.1** (+0.2)	**88.5**	76.0	86.3	**88.5** (+0.0)	82.7	70.0	80.7	**83.1** (+0.4)
$\text{RoBERTa}_{\text{large}}$	355M	**86.5**	67.2	83.8	86.2	**90.2**	76.8	88.3	90.0	85.6	70.7	84.2	**85.7** (+0.1)
$\text{DeBERTa}_{\text{xlarge}}$	750M	86.5	74.1	83.9	**87.1** (+0.6)	**91.2**	82.3	90.6	91.1	86.9	77.7	86.3	**87.3** (+0.4)

Experiments and results



(a) RTE (b) BoolQ

这两幅图表展示了P-tuning v2在不同的prompt添加深度对于两个不同任务(a) RTE (Recognizing Textual Entailment）和(b) BoolQ （Boolean Questions)的影响。

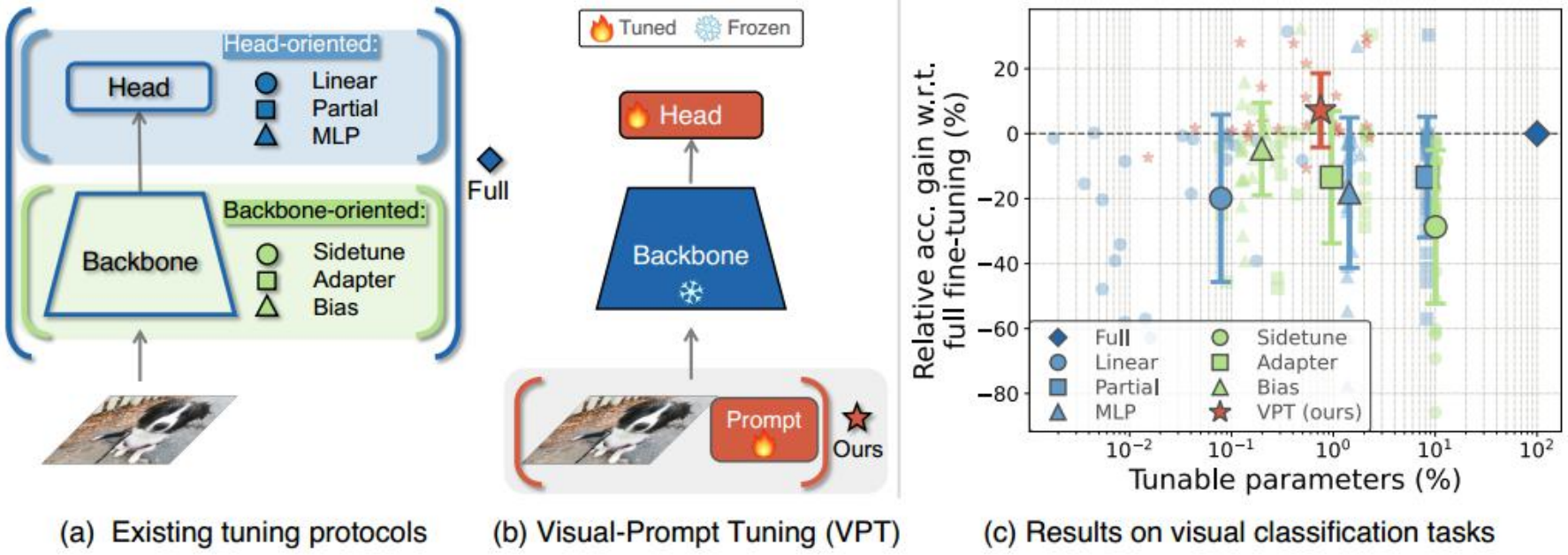
- 横轴的“ascend_order”表示添加prompts的顺序是递增的，也就是说，从较浅的层开始添加
- “descend_order”表示添加prompts的顺序是递减的，即从模型的较深层开始添加
- 递减顺序添加prompts（也就是在模型的后面几层添加）普遍比递增顺序添加（从模型的前面几层开始添加）效果要好。
- 对于RTE任务（图(a)），即使只在最后几层添加prompts（例如17-24），性能也非常接近于在所有层添加prompts的性能
- 对对于BQ，性能的提升随着添加prompts的层数的增多而增加

Visual Prompt Tuning

论文链接： https://arxiv.org/abs/2203.12119

代码链接： https://github.com/kmnp/vpt

Background and Motivation

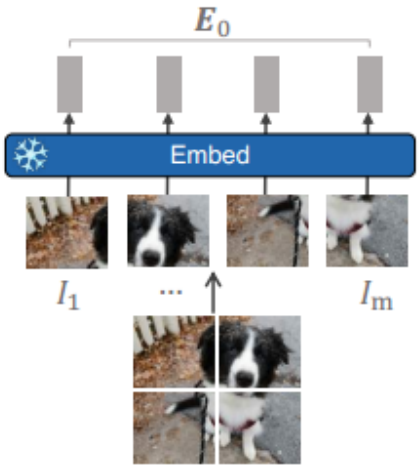


(a) Existing tuning protocols (b) Visual-Prompt Tuning (VPT) (c) Results on visual classification tasks

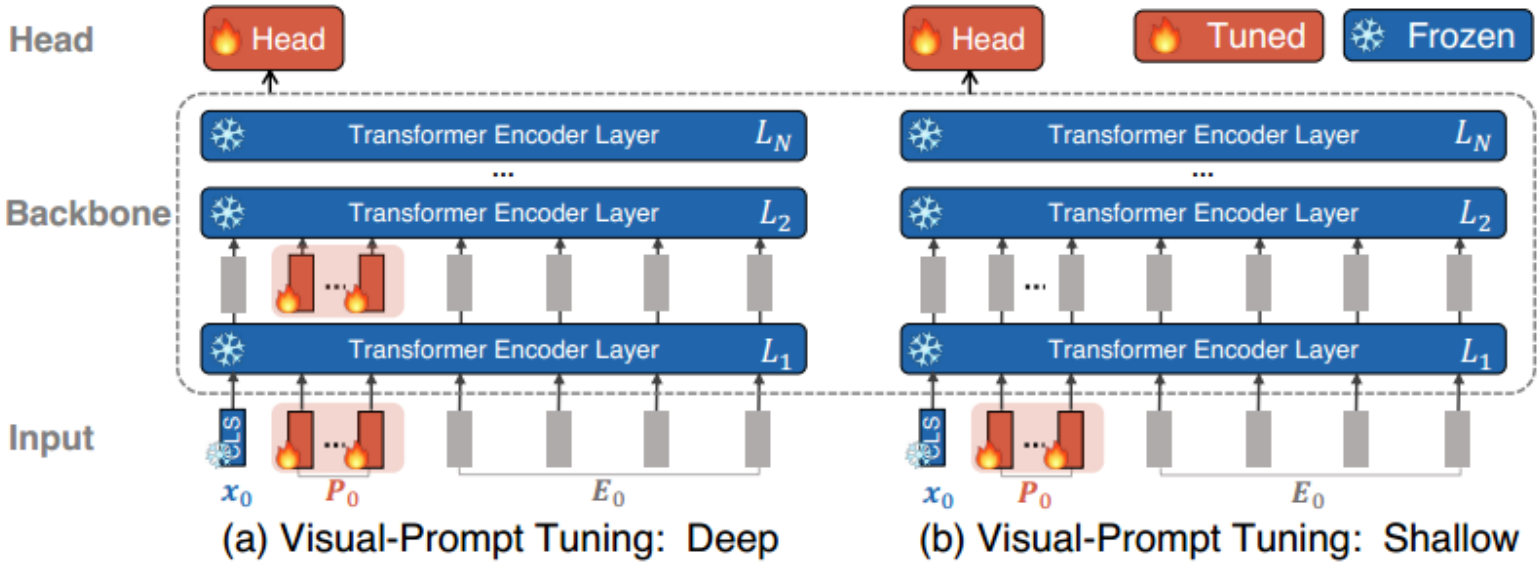
Nlp的transformer有Prompt-turning，CV的ViT有吗？有效吗？Visual Prompt Tuning保持了模型的大部分结构不变，从而减少了所需的计算资源，并且可以使模型更容易地适应多种不同的任务。

将 NLP prompt的效率带入计算机视觉领域。VPT 的性能优于完全微调，并且是使大型视觉主干适应新任务的最有效方法之一，同时使用更少的模型参数并降低每个任务的存储成本。

Method



x_i是每层的[CLS],P_i就是提示向量，E_i就是每层的图像块embedding



(a) Visual-Prompt Tuning: Deep　　(b) Visual-Prompt Tuning: Shallow

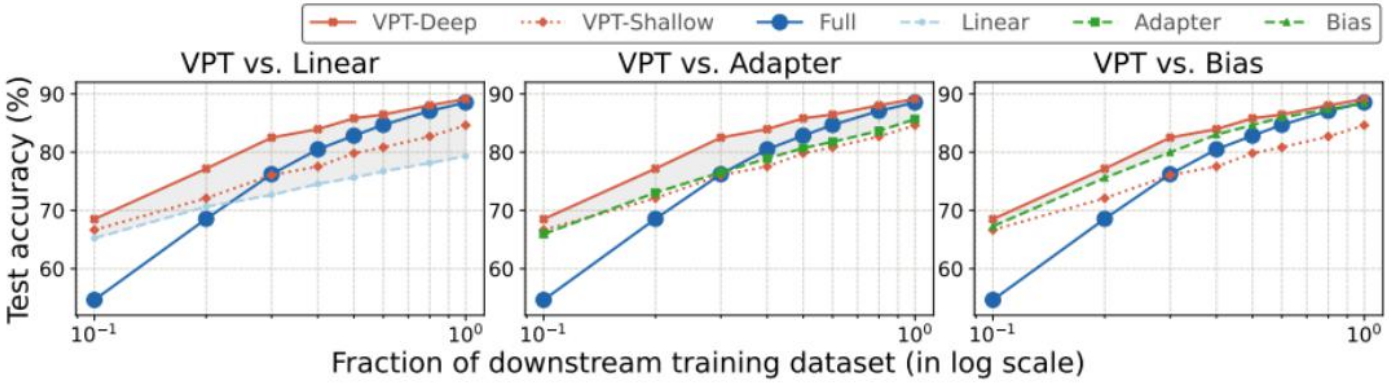
VPT的做法很直观，在embedded space中加入可学习的提示向量P，并冻结其余Encoder层。在训练时，只更新提示向量和分类头Head。

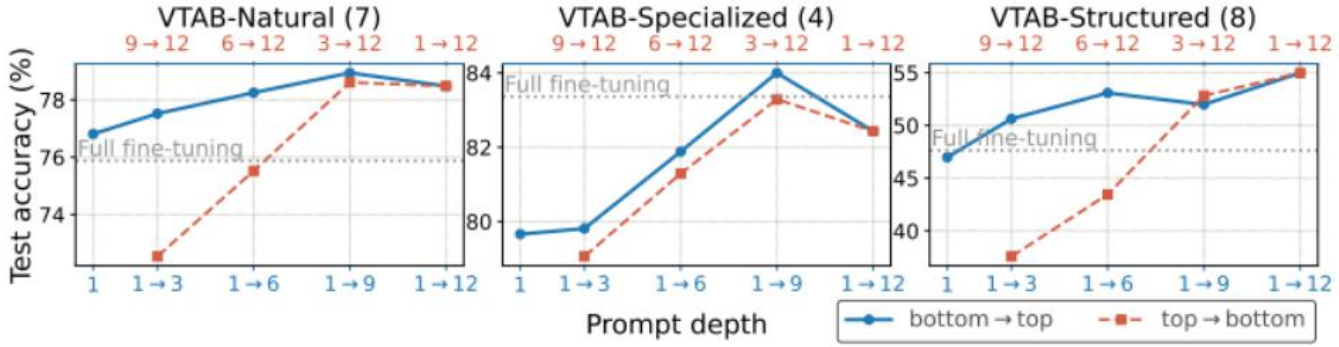
VPT Shallow :仅在第一层encoder中加入提示向量

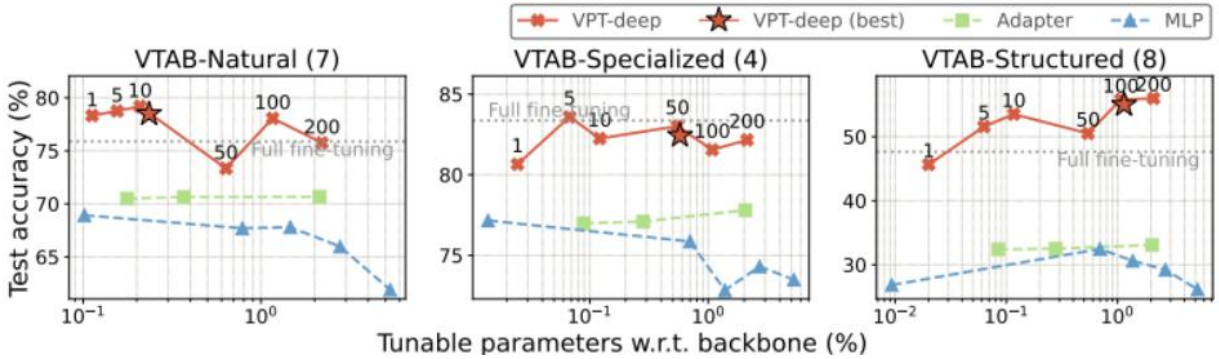
$$[\mathbf{x}_1, \mathbf{Z}_1, \mathbf{E}_1] = L_1([\mathbf{x}_0, \mathbf{P}, \mathbf{E}_0])$$
$$[\mathbf{x}_i, \mathbf{Z}_i, \mathbf{E}_i] = L_i([\mathbf{x}_{i-1}, \mathbf{Z}_{i-1}, \mathbf{E}_{i-1}])$$
$$\mathbf{y} = \mathrm{Head}(\mathbf{x}_N) \ ,$$

VPT Deep: 每一层encoder的输入中都加入提示向量

$$[\mathbf{x}_i, \_, \mathbf{E}_i] = L_i([\mathbf{x}_{i-1}, \mathbf{P}_{i-1}, \mathbf{E}_{i-1}])$$
$$\mathbf{y} = \mathrm{Head}(\mathbf{x}_N) \ .$$







横坐标是数据集规模,参数量
Tunable parameters 和 backbone 参数量的比，和prompt长度

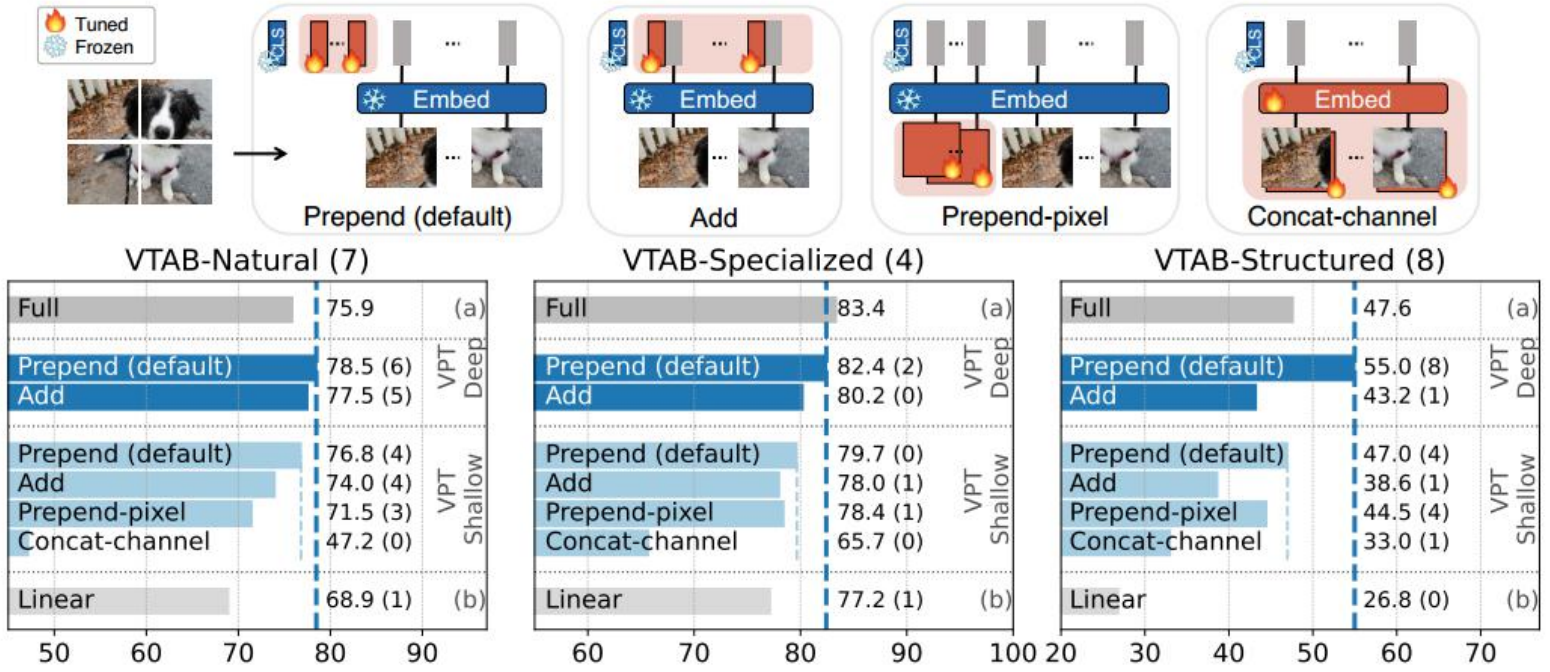


Fig. 5. Ablation on prompt location. We illustrate different location choices at top, and present the results at bottom. For easy comparison, two blue dashed lines represent the performance of the default VPT-DEEP and VPT-SHALLOW respectively

"Prepend (default)"在这三种数据集类型上通常能提供最好的性能。这可能意味着对于这个特定的模型和任务，将额外的prompt作为输入序列的前缀是一个有效的策略。

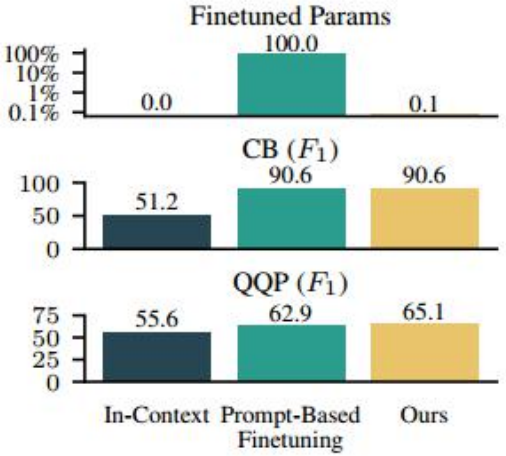
Cutting Down on Prompts and Parameters: Simple Few-Shot Learning with Language Models

论文链接：https://arxiv.org/pdf/2101.00190.pdf

代码链接：https://github.com/XiangLi1999/PrefixTuning

Background and Motivation

手动好的 prompt 的设计依赖人类直觉，即费时费力又难以复制；之前也出现了很多自动设计 prompt 的工作，但是这又引入了 prompt 自身的复杂性，比如需要大型生成模型。因此作者认为人工设计也是完全可以避免的，想找一个简单粗暴的方法。



In-Context	{What does it feel like to be on Xanax?}$_1$ and {Do 4mg Xanax bars exist?}$_2$ have different meanings. {How do you know if you're unconditionally in love with someone?}$_1$ and {How do you know if you're in love with someone and might only be denying the fact to yourself?}$_2$ have similar meanings. {Will GST affect the price level in India?}$_1$ and {Will GST effect the price level in India?}$_2$ have [MASK] meanings.
Prompt-Based Finetuning	{Will GST affect the price level in India?}$_1$? [MASK] , I want to know {Will GST effect the price level in India?}$_2$
Null Prompts (Ours)	{Will GST affect the price level in India?}$_1$ {Will GST effect the price level in India?}$_2$ [MASK]

Figure 1: **Different Methods of Few-Shot Learning.** Right: We visualize different types of prompts for QQP. We denote the input fields using curly brackets {}, the manually-written pattern using magenta, and the verbalizers using green. We show that *null prompts*, ones that do not contain training examples or task-specific patterns, can achieve competitive accuracy. Left: We compare different methods for model finetuning. Unlike standard prompt-based finetuning, we propose to update only the masked LM's bias terms (BitFit). This achieves competitive accuracy while only updating 0.1% of the parameters.

采用了一种极其简单粗暴的方法，直接将 [MASK] 添加在输入的末端，作为 prompt 模板。

Experiments and results

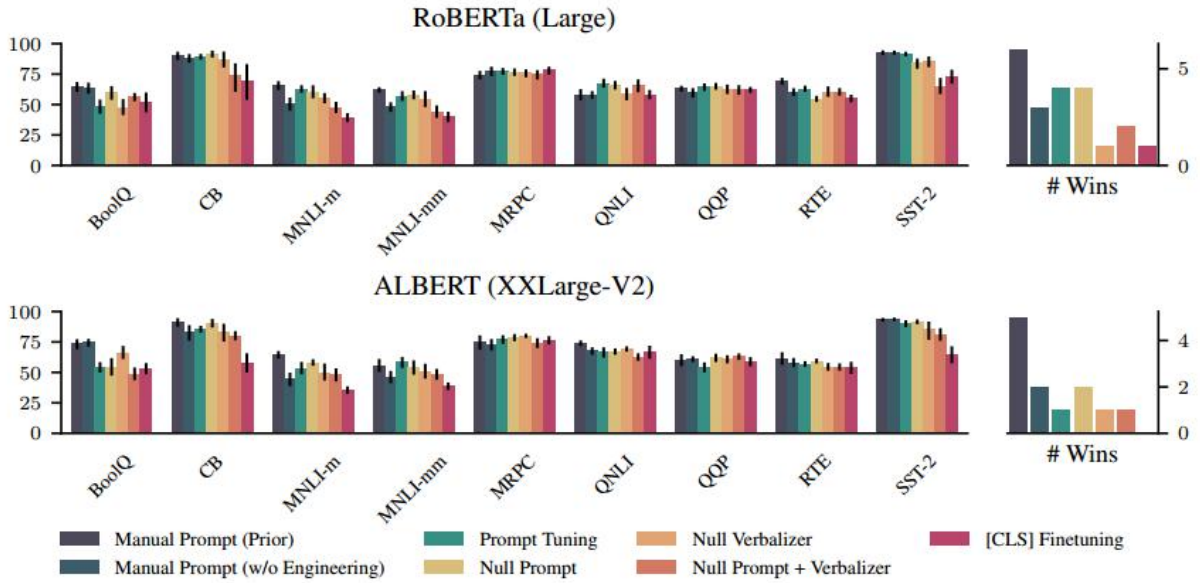


Figure 3: **Simplifying the Selection of Prompts**. We apply prompt-based finetuning in conjunction with six different types of prompts. We report accuracy or F_1 on each dataset. Manually-designed prompts from prior work achieve the best accuracy but require manual tuning on validation sets. On the other hand, null prompts and prompt tuning both perform competitively without requiring any tuning of the pattern.

Null prompts are Competitive， 这也证明了 prompt 是有效的。
而尽管人工精心设计的 prompt 依然是效果最好的，作者推荐使用 Null prompts，因为更简单，效果也不错。

Method	Finetuned Params	Prompt Design	Few-shot
AUTOPROMPT (Shin et al., 2020)	None	Learned (Discrete)	✗
Prompt Tuning (Lester et al., 2021)	Prompt Token Embeds	Learned (Continuous)	✗
OPTIPROMPT (Zhong et al., 2021)	Prompt Token Embeds	Learned (Continuous)	✗
Soft Prompts (Qin and Eisner, 2021)	All Contextualized Embeds	Learned (Continuous)	✗
GPT-3 (Brown et al., 2020)	None	Manual	✓
PET (Schick and Schütze, 2021a)	All	Manual	✓
LM-BFF (Gao et al., 2021)	All	Learned (Discrete)	✓
P-Tuning (Liu et al., 2021)	All + Prompt Token Embeds	Learned (Continuous)	✓
Null Prompts + Bitfit (Ours)	Bias Terms	None	✓

Table 1: **Overview of Existing Work on Prompting**. *Finetuned Params* indicates the parameters altered during training. *Prompt Design* indicates how prompts are created; we use *null prompts*. *Few-Shot* indicates using few-shot training and validation sets.