2.prompting

1.BitFit

1.1 背景

虽然对每个任务进行全量微调非常有效，但它也会为每个预训练任务生成一个独特的大型模型，这使得很难推断微调过程中发生了什么变化，也很难部署， 特别是随着任务数量的增加，很难维护。

理想状况下，我们希望有**一种满足以下条件的高效微调方法**：

- 到达能够匹配全量微调的效果。
- 仅更改一小部分模型参数。
- 使数据可以通过流的方式到达，而不是同时到达，便于高效的硬件部署。
- 改变的参数在不同下游任务中是一致的。

上述的问题取决于微调过程能多大程度引导新能力的学习以及暴露在预训练LM中学到的能力。

虽然，之前的高效微调方法Adapter-Tuning、Diff-Pruning也能够部分满足上述的需求。但是，作者提出了**一种参数量更小的稀疏的微调方法BitFit**，来满足上述的需求。

1.2 技术原理

BitFit（论文：**BitFit: Simple Parameter-efficient Fine-tuning or Transformer-based Masked Language-models**）是一种**稀疏的微调方法**，它**训练时只更新bias的参数或者部分bias参数。**

> **Note**
>
> **bias参数**指的就是模型中的这些偏置项

对于Transformer模型而言，**冻结大部分 transformer-encoder 参数，只更新bias参数跟特定任务的分类层参数。**涉及到的bias参数有attention模块中计算`query`,`key`,`value`跟合并多个attention结果时涉及到的bias，MLP层中的bias，Layernormalization层的bias参数。例如在 Transformer（比如 BERT）中，BitFit 会更新以下几类 bias 参数：

1. **Attention 模块中的 bias**
    - 计算 Query、Key、Value 时的 bias（虽然很多实现中 Q/K/V 的 bias 被设为 0，但有些层有）
    - 多头注意力合并后的输出层 bias
2. **MLP（前馈网络）中的 bias**
    - 每个全连接层后面的 bias
3. **LayerNorm（层归一化）中的 bias**
    - LayerNorm 有两个可学习参数：scale（γ）和 shift（β），其中 **β 就是 bias**，BitFit 会更新它

在Bert-Base/Bert-Large这种模型里，bias参数仅占模型全部参数量的0.08%～0.09%。但是通过在Bert-Large模型上基于GLUE数据集进行了 BitFit、Adapter和Diff-Pruning的效果对比发现，BitFit在参数量远小于Adapter、Diff-Pruning的情况下，效果与Adapter、Diff-Pruning相当，甚至在某些任务上略优于Adapter、Diff-Pruning。

		%Param	QNLI	SST-2	MNLI_m	MNLI_mm	CoLA	MRPC	STS-B	RTE	QQP	Avg.
	Train size		105k	67k	393k	393k	8.5k	3.7k	7k	2.5k	364k	
(V)	Full-FT†	100%	**93.5**	**94.1**	**86.5**	**87.1**	**62.8**	**91.9**	89.8	71.8	**87.6**	**84.8**
(V)	Full-FT	100%	91.7±0.1	93.4±0.2	85.5±0.4	85.7±0.4	62.2±1.2	90.7±0.3	**90.0±0.4**	**71.9±1.3**	87.5±0.4	84.1
(V)	Diff-Prune†	0.5%	**93.4**	**94.2**	**86.4**	**86.9**	63.5	91.3	89.5	71.5	**86.6**	**84.6**
(V)	BitFit	0.08%	91.4±2.4	93.2±0.4	84.4±0.2	84.8±0.1	**63.6±0.7**	**91.7±0.5**	**90.3±0.1**	**73.2±3.7**	85.4±0.1	84.2
(T)	Full-FT‡	100%	91.1	**94.9**	86.7	85.9	**60.5**	**89.3**	**87.6**	70.1	**72.1**	**81.8**
(T)	Full-FT†	100%	**93.4**	94.1	86.7	**86.0**	59.6	88.9	86.6	**71.2**	71.7	81.5
(T)	Adapters‡	3.6%	90.7	94.0	84.9	85.1	59.5	89.5	**86.9**	71.5	**71.8**	81.1
(T)	Diff-Prune†	0.5%	**93.3**	94.1	**86.4**	**86.0**	**61.1**	**89.7**	86.0	70.6	71.1	**81.5**
(T)	BitFit	0.08%	92.0	**94.2**	84.5	84.8	59.7	88.9	85.5	**72.0**	70.5	80.9

. BERT_LARGE model performance on the GLUE benchmark validation set (V) and test set (T). Lines with † and ‡ indicate results taken from Guo et al. (2020) and Houlsby et al. (2019) (respectively).

同时，通过实验结果还可以看出，**BitFit微调结果相对全量参数微调而言，只更新极少量参数的情况下，在多个数据集上都达到了不错的效果**，虽不及全量参数微调，但是远超固定全部模型参数的Frozen方式。

通过对比 **BitFit** 微调方法训练前后的偏置（bias）参数，发现**并不是所有的bias参数都发生了显著的变化。**特别是在BERT模型中，有些偏置参数（如与计算 `key` 相关的偏置）变化较小，而另一些偏置参数（如与计算 `query` 相关的偏置、或者在**前馈神经网络**（FFN）中负责特征扩展的偏置）变化较大。

	% Param	QNLI	SST-2	MNLI_m	MNLI_mm	CoLA	MRPC	STS-B	RTE	QQP	Avg.
Full-FT	100%	**90.7±0.2**	92.0±0.4	**83.5±0.1**	**83.7±0.3**	56.4±0.9	89.0±1.0	88.9±0.7	70.5±0.6	**87.1±0.1**	82.3
BitFit	0.09%	90.2±0.2	**92.1±0.3**	81.4±0.2	82.2±0.2	**58.8±0.5**	**90.4±0.5**	**89.2±0.2**	**72.3±0.9**	84.0±0.2	**82.4**
$\mathbf{b}_{m2}, \mathbf{b}_q$	0.04%	89.4±0.1	91.2±0.2	80.4±0.2	81.5±0.2	57.4±0.8	89.0±0.2	88.4±0.1	68.6±0.6	83.7±0.2	81.1
$\mathbf{b}_{m2}$	0.03%	88.9±0.1	91.1±0.3	79.9±0.3	80.7±0.2	54.9±0.9	87.9±0.6	88.2±0.1	66.8±0.6	82.1±0.4	80.0
$\mathbf{b}_q$	0.01%	86.8±0.1	89.6±0.2	74.4±0.3	75.7±0.2	49.1±1.5	84.4±0.2	85.6±0.1	61.4±1.1	80.6±0.4	76.6
Frozen	0.0%	68.7±0.3	81.7±0.1	42.4±0.1	43.8±0.1	31.9±1.1	81.1±0.1	71.4±0.1	56.9±0.4	62.4±0.2	62.1
rand uniform	0.09%	87.8±0.3	90.5±0.3	78.3±0.3	78.8±0.2	54.1±1.0	84.3±0.3	87.2±0.4	62.9±0.9	82.4±0.3	78.5
rand row/col	0.09%	88.4±0.2	91.0±0.3	79.4±0.3	80.1±0.3	53.4±0.6	88.0±0.7	87.9±0.2	65.1±0.7	82.3±0.2	79.5

Fine-tuning using a subset of the bias parameters. Reported results are for the BERT_BASE model

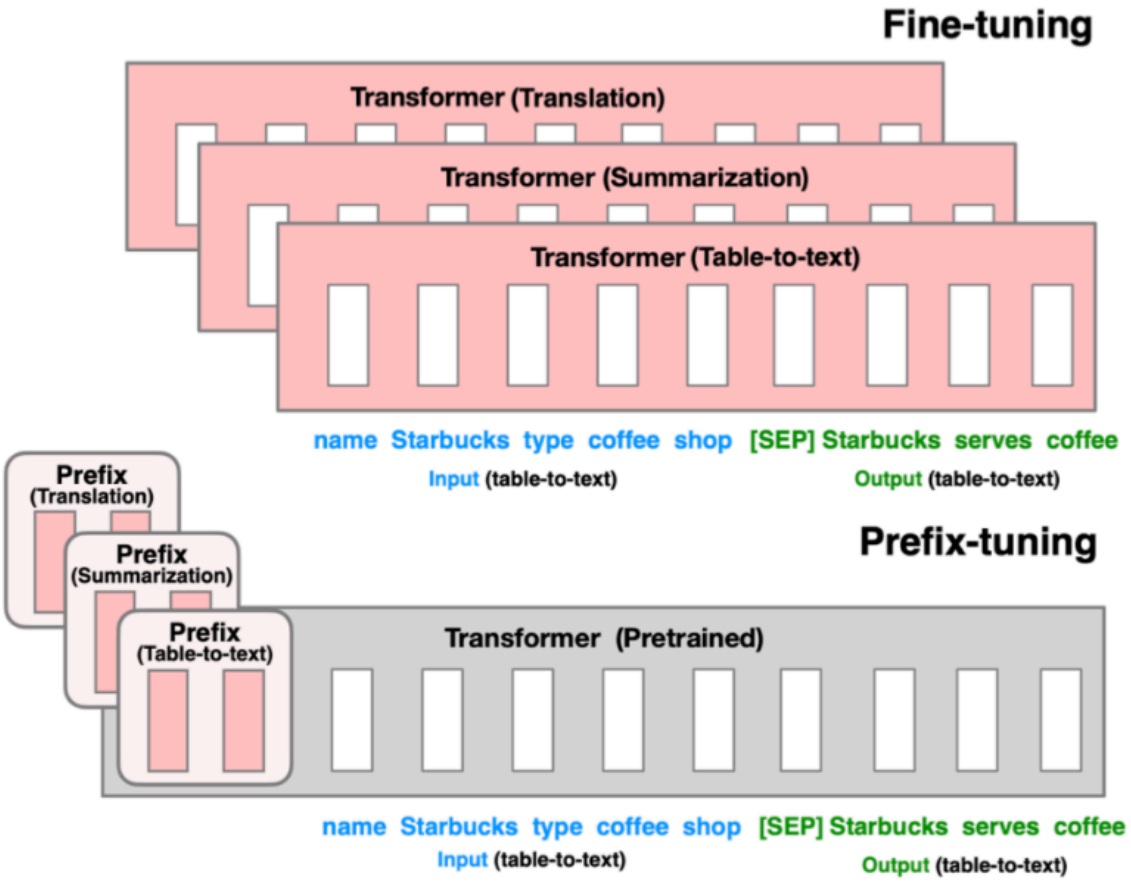
2.Prefix Tuning

2.1 背景

在Prefix Tuning之前的工作主要是人工设计离散的模版或者自动化搜索离散的模版。对于人工设计的模版，模版的变化对模型最终的性能特别敏感，加一个词、少一个词或者变动位置都会造成比较大的变化。而对于自动化搜索模版，成本也比较高；同时，以前这种离散化的token搜索出来的结果可能并不是最优的。

除此之外，传统的微调范式利用预训练模型去对不同的下游任务进行微调，对每个任务都要保存一份微调后的模型权重，一方面微调整个模型耗时长；另一方面也会占很多存储空间。

基于上述两点，Prefix Tuning提出**固定预训练LM，为LM添加可训练，任务特定的前缀，** 这样就可以为不同任务保存不同的前缀，微调成本也小；同时，这种Prefix实际就是连续可微的Virtual Token（Soft Prompt/Continuous Prompt），相比离散的Token，更好优化，效果更好。



2.2 技术原理

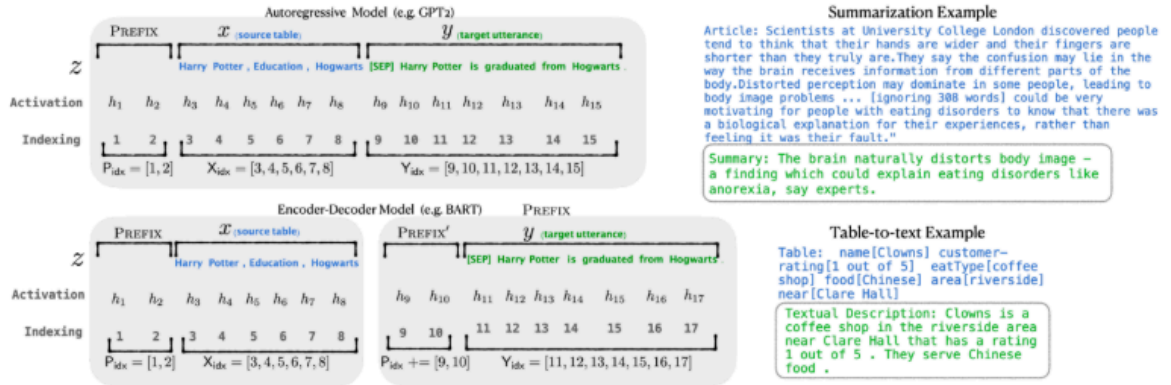
Prefix Tuning（论文：**Prefix-Tuning: Optimizing Continuous Prompts for Generation**），**在输入token之前构造一段任务相关的virtual tokens作为Prefix，然后训练的时候只更新Prefix部分的参数，而PLM中的其他部分参数固定。**

针对不同的模型结构，需要构造不同的Prefix。

- **针对自回归架构模型**：**在句子前面添加前缀**，得到 `z = [PREFIX; x; y]`，合适的上文能够在固定 LM 的情况下去引导生成下文（比如：GPT3的上下文学习）。
- **针对编码器-解码器架构模型**：**Encoder和Decoder都增加了前缀**，得到 `z = [PREFIX; x; PREFIX0; y]`。Encoder端增加前缀是为了引导输入部分的编码，Decoder 端增加前缀是为了引导后续token的生成。

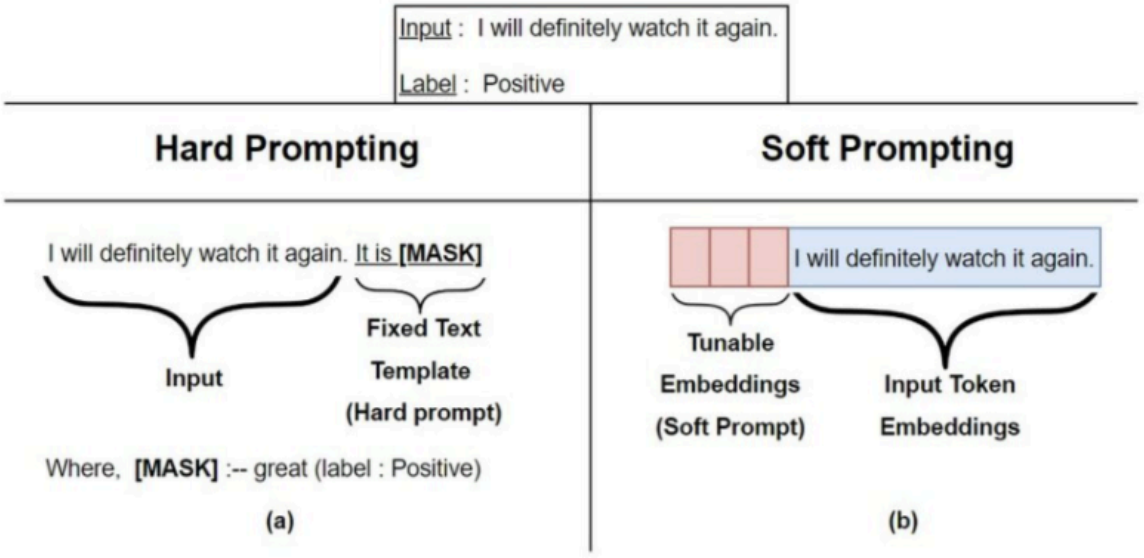
Encoder Prefix：加在输入 `x` 前面 → 帮助更好地"理解"输入

Decoder Prefix：加在输出 `y` 前面 → 帮助更好地"生成"答案



An annotated example of prefix-tuning using an autoregressive LM (top) and an encoder-decoder model (bottom). The prefix activations $\forall i \in \mathsf{P}_{\mathsf{idx}}, h_i$ are drawn from a trainable matrix P_θ. The remaining activations are computed by the Transformer.

该方法其实和构造Prompt类似，只是Prompt是人为构造的“显式”的提示，并且无法更新参数，而Prefix则是可以学习的“隐式”的提示。



同时，**为了防止直接更新Prefix的参数导致训练不稳定和性能下降的情况，在Prefix层前面加了MLP结构，训练完成后，只保留Prefix的参数。**

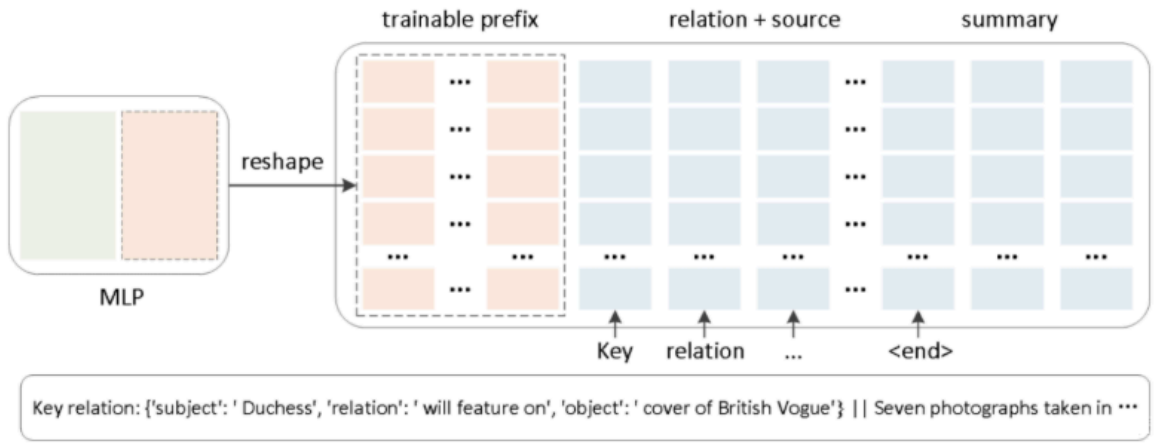
这些 Prefix 向量一开始是随机初始化的，而 Transformer 的其余部分（比如 BERT、GPT）是**在大量文本上预训练好的**，内部已经形成了非常精细的数值分布（比如注意力权重、激活值的范围等），如果直接让随机的 Prefix 向量“硬闯”进这个精密系统：

- 它们的数值范围可能和预训练模型期望的输入**严重不匹配**（比如模型习惯输入在 [-1, 1]，但随机初始化在 [-0.5, 0.5] 或更大）
- 这会导致**注意力机制混乱**，梯度爆炸或消失
- 训练过程就容易**震荡、不稳定，甚至性能比不用还差**

具体流程如下：

1. 定义一个**可学习的轻量级参数矩阵** `P`（比如形状是 `[prefix_len, hidden_size]`），这是“基础向量”，随机初始化。
2. 把 `P` 输入到一个**小型 MLP**（比如 Linear → ReLU → Linear）中。
3. MLP 的输出就是**最终用于模型的 Prefix 向量**（会被插入到 Key/Value 中）。
4. **训练时，只更新 `P` 和 MLP 的参数**，大模型冻结。

5. **训练完成后，把 MLP 的输出"固化"下来**——也就是直接保存 MLP(P) 的结果作为最终的 Prefix，**丢掉 MLP 本身。**

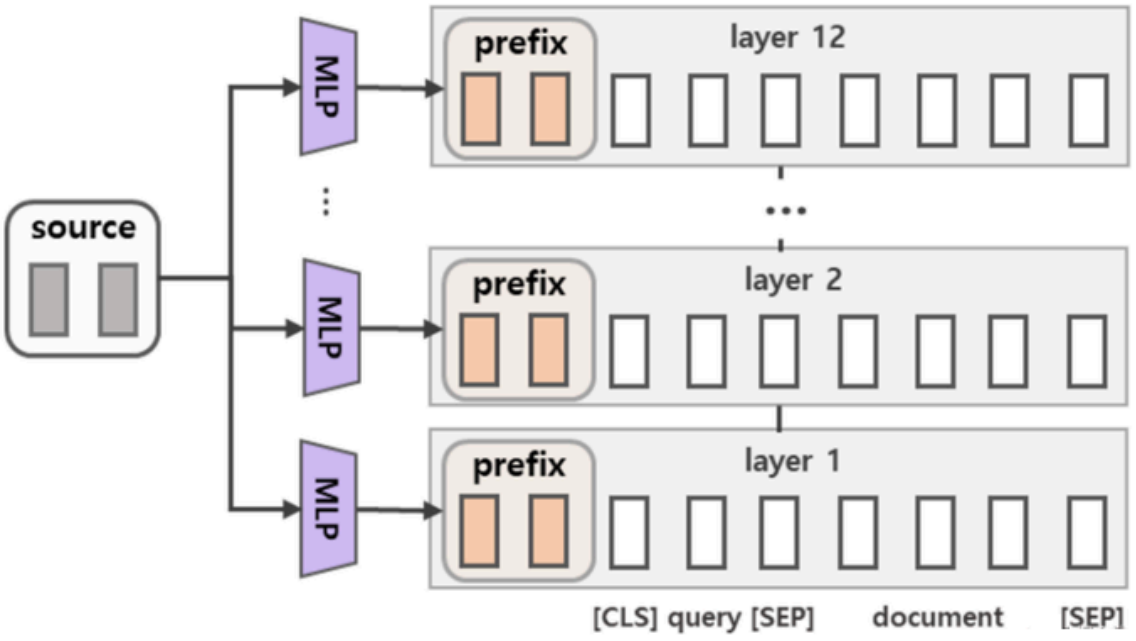


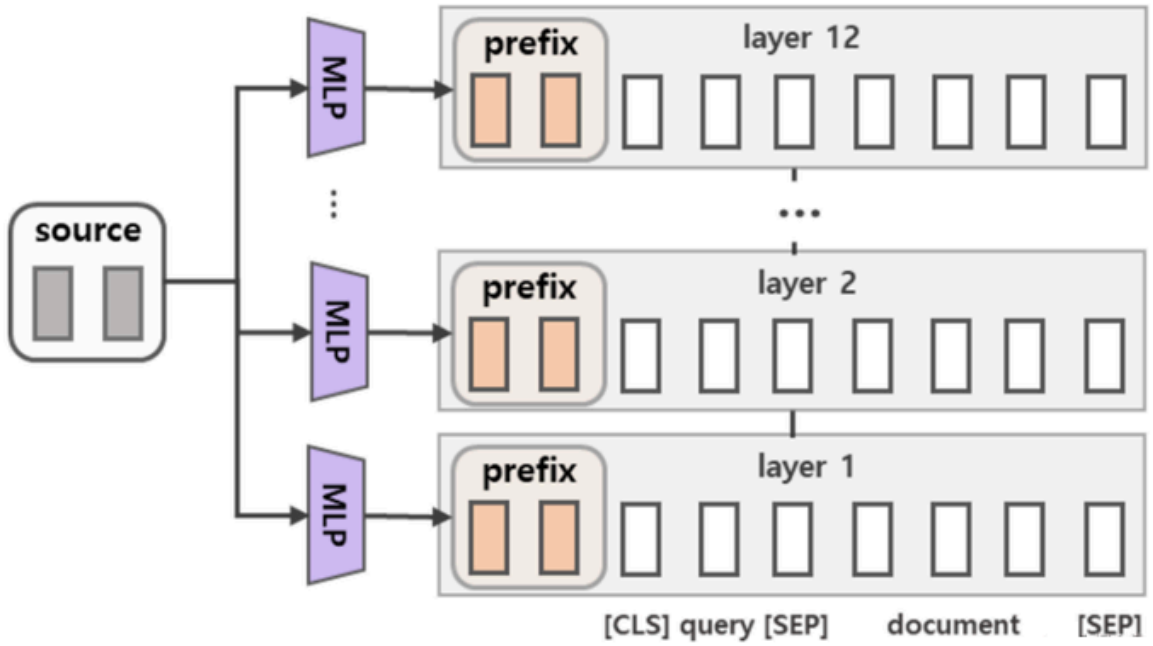
除此之外，通过消融实验证实，只调整embedding层的表现力不够，将导致性能显著下降，因此，在每层都加了prompt的参数，改动较大。

- **Prompt Tuning**：只在第0层（输入层）加软提示 → 信息容易丢失
- **Prefix Tuning**：在第1~L层（每一层）都加软提示 → 信息全程陪伴

总结对比表

方法	提示位置	可训练参数位置	表达能力	实际效果
Prompt Tuning	仅输入 embedding 层	只有开头几个 token 的 embedding	较弱	小模型还行，大模型效果差
Prefix Tuning	**每一层的 Key/Value 中**	每层都有独立的 prefix 向量	很强	大模型也能高效微调





另外，实验还对比了位置对于生成效果的影响，Prefix-tuning也是要略优于Infix-tuning的。其中，Prefix-tuning形式为 `[PREFIX; x; y]`，Infix-tuning形式为 `[x; INFIX; y]`。

	E2E				
	BLEU	NIST	MET	ROUGE	CIDEr
PREFIX	69.7	8.81	46.1	71.4	2.49
Embedding-only: EMB-{PrefixLength}					
EMB-1	48.1	3.33	32.1	60.2	1.10
EMB-10	62.2	6.70	38.6	66.4	1.75
EMB-20	61.9	7.11	39.3	65.6	1.85
Infix-tuning: INFIX-{PrefixLength}					
INFIX-1	67.9	8.63	45.8	69.4	2.42
INFIX-10	67.2	8.48	45.8	69.9	2.40
INFIX-20	66.7	8.47	45.8	70.0	2.42

Intrinsic evaluation of Embedding-only (§7.2) and Infixing (§7.3). Both Embedding-only ablation and Infix-tuning underperforms full prefix-tuning.

3.Prompt Tuning

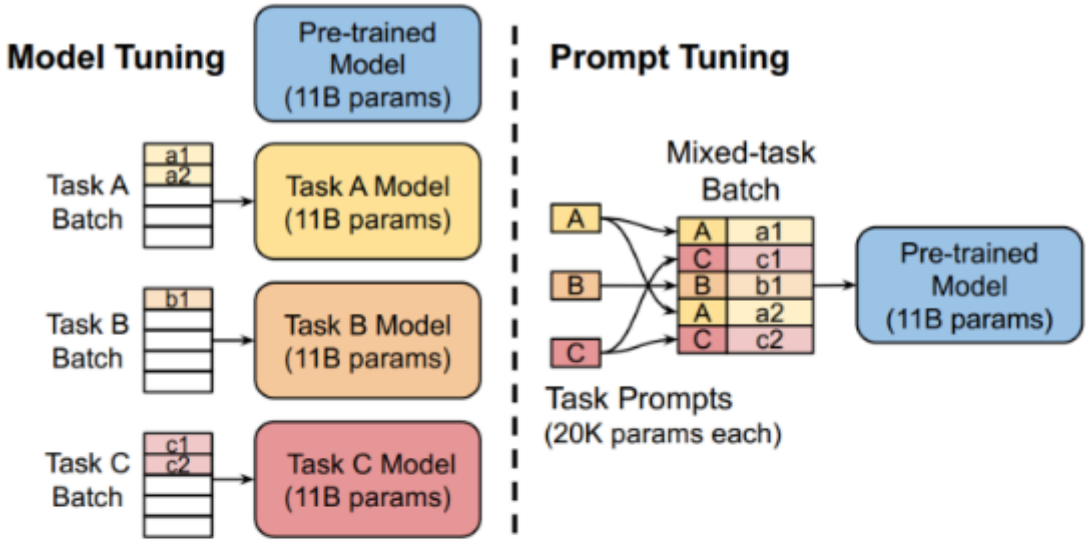
3.1 背景

大模型全量微调对每个任务训练一个模型，开销和部署成本都比较高。同时，离散的prompts（指人工设计prompts提示语加入到模型）方法，成本比较高，并且效果不太好。

基于此，作者提出了Prompt Tuning，**通过反向传播更新参数来学习prompts，而不是人工设计prompts；同时冻结模型原始权重，只训练prompts参数，** 训练完以后，用同一个模型可以做多任务推理。

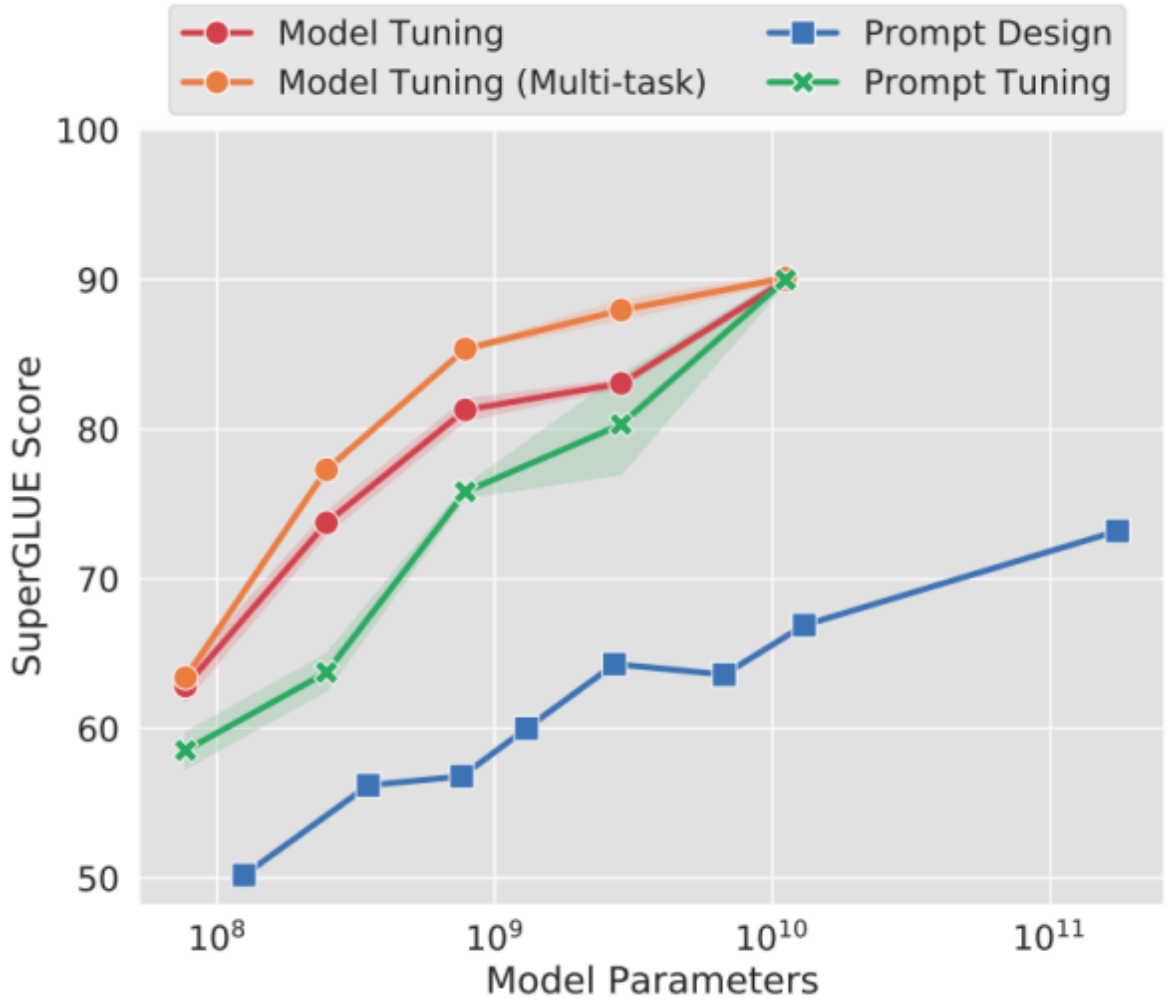
3.2 技术原理

Prompt Tuning（论文：**The Power of Scale for Parameter-Efficient Prompt Tuning**），该方法可以看作是Prefix Tuning的简化版本，它给**每个任务定义了自己的Prompt，然后拼接到数据上作为输入，但只在输入层加入prompt tokens**，并且不需要加入 MLP 进行调整来解决难训练的问题。

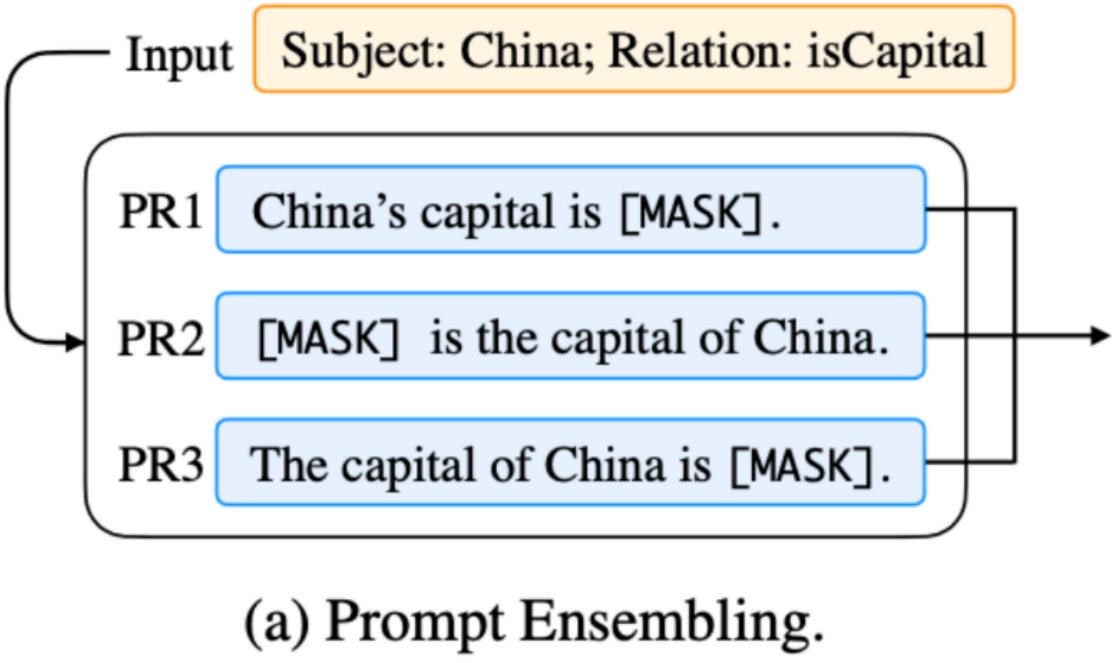


Model tuning requires making a task-specific copy of the entire pre-trained model for each downstream task and inference must be performed in separate batches. **Prompt tuning** only requires storing a small task-specific prompt for each task, and enables mixed-task inference using the original pre-trained model. With a T5 "XXL" model, each copy of the tuned model requires 11 billion parameters. By contrast, our tuned prompts would only require 20,480 parameters per task—a reduction of *over five orders of magnitude*—assuming a prompt length of 5 tokens

通过实验发现，随着预训练模型参数量的增加，Prompt Tuning的方法会逼近全参数微调的结果。



同时，Prompt Tuning 还提出了 Prompt Ensembling，也就是**在一个批次（Batch）里同时训练同一个任务的不同 prompt（即采用多种不同方式询问同一个问题）**，这样相当于训练了不同模型，比模型集成的成本小多了。



(a) Prompt Ensembling.

内部表示（hidden states）和注意力行为

方法	内部表示特点	注意力行为
Prompt Tuning	提示 token 的 embedding 会随着层数加深逐渐被“同化”或“遗忘”	注意力可能很快忽略开头的 prompt tokens（尤其当输入很长时）
Prefix Tuning	每一层都强制注入任务信息，hidden states 始终带有任务 bias	注意力机制**主动关注 prefix 部分**

为什么 Prompt Tuning 不需要 MLP，而 Prefix Tuning 需要?

- **Prompt Tuning** 的软提示是**作为普通 token 输入 embedding 层的**，
  所以它的数值范围天然和词 embedding 一致（比如都在 [-2, 2]），**不会造成数值冲突**，训练相对稳定。
- **Prefix Tuning** 的 prefix 是**直接拼接到 Key/Value 矩阵中的**，
  而 Key/Value 经过线性变换后，数值分布和原始 embedding 不同。
  如果随机初始化 prefix，容易和预训练分布不匹配 → **需要 MLP 做适配**（就像我们之前聊的“雕塑家”比喻）。

> **Note**
>
> **Prefix Tuning 确实没有改变输入序列本身，而是通过在 Transformer 的每一层中，向注意力机制的 Key（K）和 Value（V）矩阵中“注入”可学习的前缀向量，从而直接改变注意力的计算过程。**

4.P-Tuning

4.1 背景

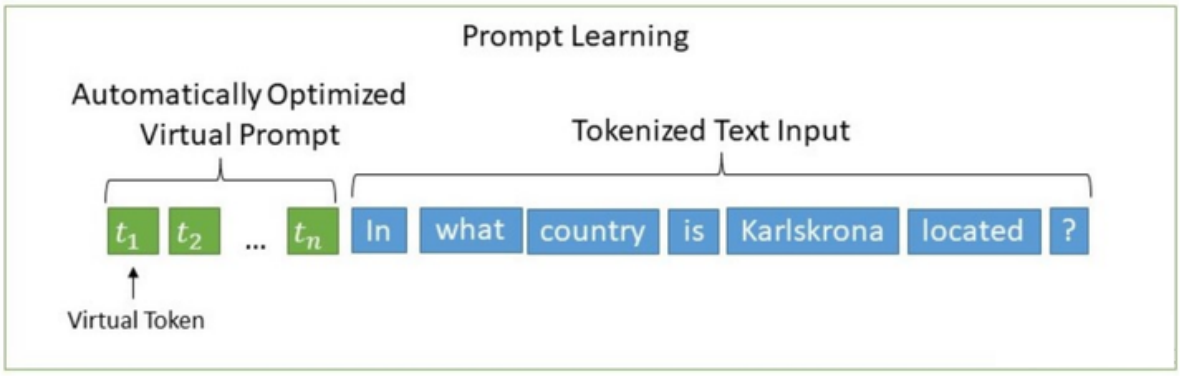
该方法的提出主要是为了解决这样一个问题：**大模型的Prompt构造方式严重影响下游任务的效果。**比如：GPT-3采用人工构造的模版来做上下文学习（in context learning），但人工设计的模版的变化特别敏感，加一个词或者少一个词，或者变动位置都会造成比较大的变化。

Prompt	P@1
[X] is located in [Y]. *(original)*	31.29
[X] is located in which country or state? [Y].	19.78
[X] is located in which country? [Y].	31.40
[X] is located in which country? In [Y].	51.08

Case study on LAMA-TREx P17 with bert-base-cased. A single-word change in prompts could yield a drastic difference.

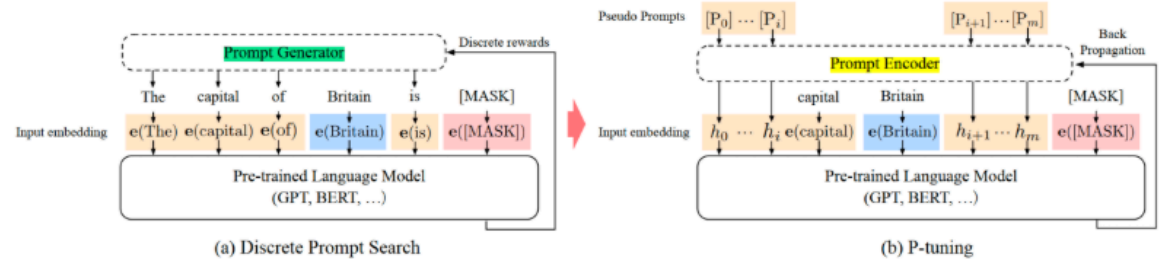
同时，近来的自动化搜索模版工作成本也比较高，以前这种离散化的token的搜索出来的结果可能并不是最优的，导致性能不稳定。

基于此，作者提出了P-Tuning，设计了一种**连续可微的virtual token**（同Prefix-Tuning类似）。



4.2 技术原理

P-Tuning（论文：**GPT Understands, Too**），该方法**将Prompt转换为可以学习的Embedding层，并用MLP+LSTM的方式来对Prompt Embedding进行一层处理。**



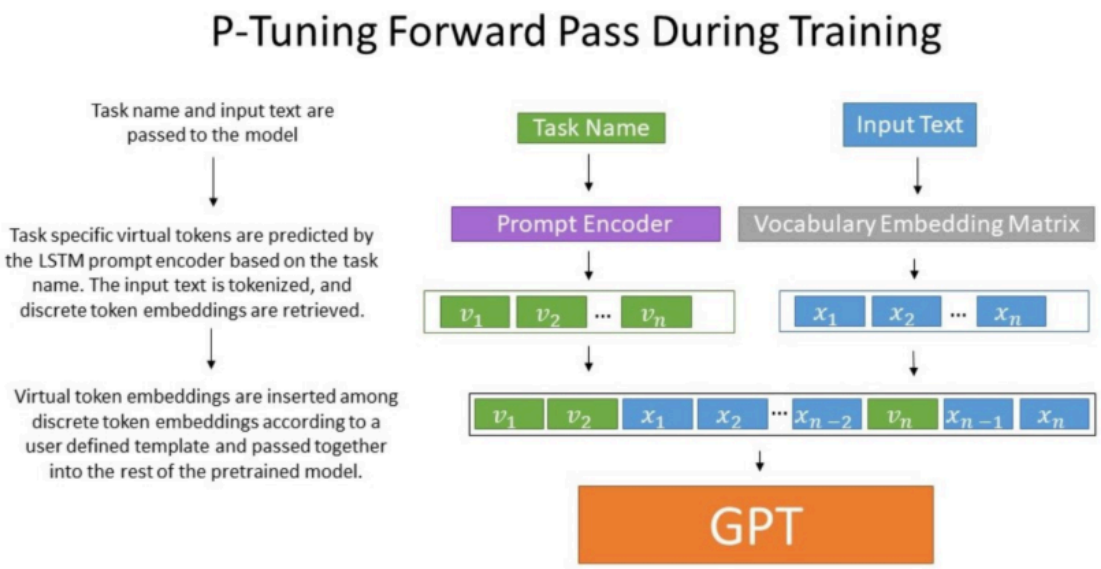
An example of prompt search for "The capital of Britain is [MASK]". Given the context (blue zone, "Britain") and target (red zone, "[MASK]"), the orange zone refer to the prompt tokens. In (a), the prompt generator only receives discrete rewards; on the contrary, in (b) the pseudo prompts and prompt encoder can be optimized in a differentiable way. Sometimes, adding few task-related anchor tokens (such as "capital" in (b)) will bring further improvement.

相比Prefix Tuning，P-Tuning加入的可微的virtual token，**但仅限于输入层，没有在每一层都加；另外，virtual token的位置也不一定是前缀，插入的位置是可选的。**这里的出发点实际是把传统人工设计模版中的真实token替换成可微的virtual token。

> **Note**
>
> **P-Tuning 把 prompt 看作一个"可学习的句子模板"，其中某些位置是固定的自然语言词（比如 "It was"），些位置是可学习的虚拟 token（称为 prompt tokens），并用一个小型神经网络（MLP + LSTM）来优化这些虚拟 token 的表示，让它们协同工作**
>
> **举个具体例子（情感分析）：**
>
> - **模板设计**：
>   `"It was [x1] [x2]. [Sentence]"`
>   - "It was" 和 "." 是**固定的真实词**（frozen embedding）
>   - `[x1]`, `[x2]` 是**可学习的虚拟 token**
> - **输入序列**：`[It, was, x1, x2, ., I, love, this, movie]`
> - **关键创新**：
>   x1 和 x2 的 embedding **不是直接优化**，而是：
>   1. 先有一个可学习的"基础向量" `E = [e1, e2]`（随机初始化）
>   2. 把 E 输入到一个 **小型 LSTM（或 MLP）** 中
>   3. LSTM 输出：`[h1, h2]` → 这才是真正的 prompt embedding

```
E = nn.Parameter(torch.randn(2, hidden_size))   # 可学习基础向量
lstm = nn.LSTM(hidden_size, hidden_size, batch_first=True)
prompt_embeds, _ = lstm(E.unsqueeze(0))         # 输出有上下文的 prompt
```

- **训练时**：只更新 `E` 和 LSTM 的参数，大模型冻结



经过预训练的LM的词嵌入已经变得高度离散，如果随机初始化virtual token，容易优化到局部最优值，而这些virtual token理论是应该有相关关联的。因此，作者通过实验发现**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virtual token以后，再输入到模型。

> **Note**
>
> **为什么加 LSTM/MLP 有效?**
>
> 1. **引入顺序建模能力**
>   LSTM 能让 x1 和 x2 **有语法关系**：x2 的表示依赖于 x1，就像真实语言中“really”影响“good”。
> 2. **提升表达能力**
>   相比孤立向量，LSTM 输出的 prompt 更像“一个短语”，而不是“两个随机点”。
> 3. **训练更稳定**
>   MLP/LSTM 起到“正则化”作用，避免 prompt embedding 过拟合或震荡。

从对比实验证实看出，P-Tuning获得了与全参数一致的效果。甚至在某些任务上优于全参数微调。

并且在实验中还发现，相同参数规模，如果进行全参数微调，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。

5.P-Tuning v2

5.1 背景

之前的Prompt Tuning和P-Tuning等方法存在两个主要的问题：

第一，**缺乏模型参数规模和任务通用性。**

- **缺乏规模通用性**：Prompt Tuning论文中表明当模型规模超过100亿个参数时，提示优化可以与全量微调相媲美。但是对于那些较小的模型（从100M到1B），提示优化和全量微调的表现有很大差异，这大大限制了提示优化的适用性。

- **缺乏任务普遍性**：尽管Prompt Tuning和P-tuning在一些 NLU(自然语言理解) 基准测试中表现出优势，但提示调优对硬序列标记任务（即序列标注）的有效性尚未得到验证。

第二，**缺少深度提示优化**，在Prompt Tuning和P-tuning中，连续提示只被插入transformer第一层的输入embedding序列中，在接下来的transformer层中，插入连续提示的位置的embedding是由之前的transformer层计算出来的，这可能导致两个可能的优化挑战。

- 由于序列长度的限制，可调参数的数量是有限的。
- 输入embedding对模型预测只有相对间接的影响。

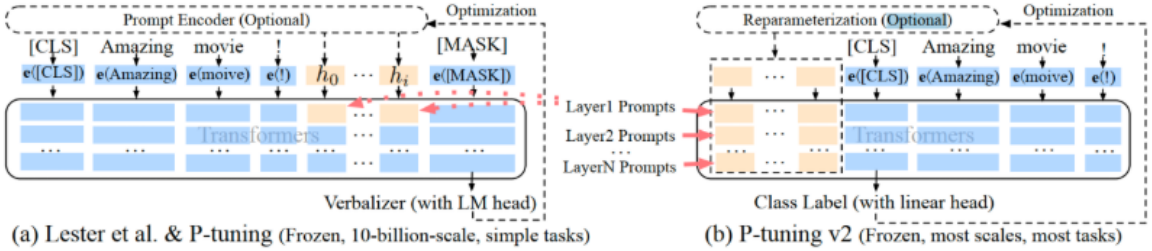
考虑到这些问题，作者提出了Ptuning v2，它**利用深度提示优化（如：Prefix Tuning），对Prompt Tuning和P-Tuning进行改进，作为一个跨规模和NLU任务的通用解决方案。**

5.2 技术原理

P-Tuning v2（论文： **P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks**），该方法**在每一层都加入了Prompts tokens作为输入，而不是仅仅加在输入层**，这带来两个方面的好处：

- 更多可学习的参数（从P-tuning和Prompt Tuning的0.01%增加到0.1%-3%），同时也足够参数高效。
- 加入到更深层结构中的Prompt能给模型预测带来更直接的影响。

> **Note**
>
> > **P-Tuning v2 的实现方式，本质上和 Prefix Tuning 几乎完全相同——**
> > **它不是简单地在输入层加 token，而是将可学习的 prompt 向量直接注入到 Transformer 每一层的 Key 和 Value 矩阵中，从而修改了注意力计算。**
>
> 换句话说：**它没有改变输入序列，** **但它修改了每一层的 K/V 矩阵。**
>
> 核心机制：
>
> 对于 每一层 的 Transformer：
>
> - 在 **Key（K）和 Value（V）** 矩阵的**前面拼接**一段可学习的向量
> - 这些向量就是 **per-layer prompts**
> - Query（Q）仍然只来自原始输入（不加 prompt）
>
> ```
> # 第 l 层的实际计算：
> K = torch.cat([prompt_key[l], K_from_input], dim=1)   # (batch, prompt_len + seq_len, hidden)
> V = torch.cat([prompt_value[l], V_from_input], dim=1)
> Q = Q_from_input                                      # (batch, seq_len, hidden)
> ```
>
> - 注意力权重 = `softmax(Q @ K^T / sqrt(d))`
> - 输出 = `Attention @ V`



(a) Lester et al. & P-tuning (Frozen, 10-billion-scale, simple tasks) (b) P-tuning v2 (Frozen, most scales, most tasks)

From Lester et al. (2021) & P-tuning to P-tuning v2. Orange blocks (i.e., $h_0, ..., h_i$) refer to trainable prompt embeddings; blue blocks are embeddings stored or computed by frozen pre-trained language models.

具体做法基本同Prefix Tuning，可以看作是将文本生成的Prefix Tuning技术适配到NLU任务中，然后做了一些改进：

- **移除重参数化的编码器。**以前的方法利用重参数化功能来提高训练速度和鲁棒性（如：Prefix Tuning中的MLP、P-Tuning中的LSTM））。在 P-tuning v2 中，作者发现重参数化的改进很小，尤其是对于较小的模型，同时还会影响模型的表现。
  (发现在 NLU 任务中，直接优化 prompt 向量效果更好)
- **针对不同任务采用不同的提示长度。**提示长度在提示优化方法的超参数搜索中起着核心作用。在实验中，我们发现不同的理解任务通常用不同的提示长度来实现其最佳性能，这与Prefix-Tuning中的发现一致，**不同的文本生成任务可能有不同的最佳提示长度。**
- **引入多任务学习。**先在多任务的Prompt上进行预训练，然后再适配下游任务。多任务学习对我们的方法来说是可选的，但可能是相当有帮助的。一方面，连续提示的随机惯性给优化带来了困难，这可以通过更多的训练数据或与任务相关的无监督预训练来缓解；另一方面，连续提示是跨任务和数据集的特定任务知识的完美载体。我们的实验表明，在一些困难的序列任务中，多任务学习可以作为P-tuning v2的有益补充。

  (先在多个 NLU 任务上联合训练 prompt,再迁移到下游任务 → 缓解小数据优化难的问题)
- **回归传统的分类标签范式，而不是映射器。**标签词映射器（Label Word Verbalizer）一直是提示优化的核心组成部分，它将one-hot类标签变成有意义的词，以利用预训练语言模型头。尽管它在few-shot设置中具有潜在的必要性，但在全数据监督设置中，Verbalizer（标签词映射器）并不是必须的。它阻碍了提示调优在我们需要无实际意义的标签和句子嵌入的场景中的应用。因此，P-Tuning v2回归传统的CLS标签分类范式，采用随机初始化的分类头（Classification Head）应用于tokens之上，以增强通用性，可以适配到序列标注任务。

> **Note**
>
> 举个Verbalizer具体例子：
>
> | 任务标签 | VERBALIZER映射的值（LABEL WORDS） |
> | --- | --- |
> | positive | ["great", "good", "excellent"] |
> | negative | ["bad", "terrible", "awful"] |
>
> 训练/推理时：
>
> - 模型看到：`"It was [MASK]."`
> - 预测 `[MASK]` 是 “great”
> - Verbalizer 查表：**“great” ∈ positive 的词集 → 输出 “positive”**
>
> 完整流程示例：情感分析任务
>
> 1. **输入**：`"[CLS] I love NLP [SEP]"`
> 2. 模型内部：

- 每一层的 K/V 前面拼接可学习 prefix
- 信息流经 12 层，被 prefix 持续引导

3. **取 [CLS] 的最终 hidden state**：`h_cls`（768 维）
4. **分类头**：`logits = W · h_cls + b`
5. **损失函数**：交叉熵（logits vs 真实标签 "positive"）
6. **反向传播**：只更新 prefix 向量 + W, b

输出：直接是 “positive” 或 “negative”，**无需 Verbalizer**

对比总结表

特性	PREFIX TUNING	P-TUNING V2
核心机制	每层 K/V 加 prefix	完全相同
主要任务	NLG（生成）	**NLU（理解） + 生成**
输出方式	Verbalizer + LM head	**分类头**（通用）
支持 NER/序列标注?	否	是
重参数化（MLP）	用	不用
多任务 prompt 预训练	未设计	支持
模型适配	T5, GPT	**BERT, RoBERTa, T5, GPT...**
通用性	中（限生成）	高（全任务）

论文中展示了P-tuning v2在不同模型规模下的表现。对于简单的NLU任务，如SST-2（单句分类），Prompt Tuning和P-Tuning在较小的规模下没有显示出明显的劣势。但是当涉及到复杂的挑战时，如：自然语言推理（RTE）和多选题回答（BoolQ），它们的性能会非常差。相反，P-Tuning v2在较小规模的所有任务中都与微调的性能相匹配。并且，P-tuning v2在RTE中的表现明显优于微调，特别是在BERT中。

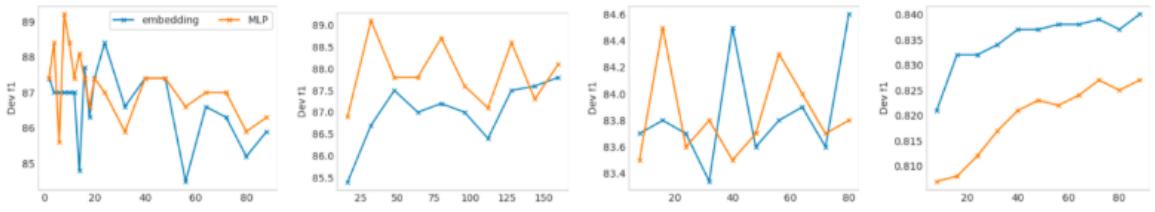
	#Size	BoolQ			CB			COPA			MultiRC (F1a)		
		FT	PT	PT-2	FT	PT	PT-2	FT	PT	PT-2	FT	PT	PT-2
$\text{BERT}_{\text{large}}$	335M	**77.7**	67.2	75.8	**94.6**	80.4	**94.6**	69.0	55.0	**73.0**	70.5	59.6	**70.6**
$\text{RoBERTa}_{\text{large}}$	355M	**86.9**	62.3	84.8	98.2	71.4	**100**	**94.0**	63.0	93.0	**85.7**	59.9	82.5
$\text{GLM}_{\text{xlarge}}$	2B	**88.3**	79.7	87.0	**96.4**	76.4	**96.4**	**93.0**	92.0	91.0	84.1	77.5	**84.4**
$\text{GLM}_{\text{xxlarge}}$	10B	88.7	**88.8**	**88.8**	**98.7**	98.2	96.4	**98.0**	**98.0**	**98.0**	**88.1**	86.1	**88.1**

	#Size	ReCoRD (F1)			RTE			WiC			WSC		
		FT	PT	PT-2	FT	PT	PT-2	FT	PT	PT-2	FT	PT	PT-2
$\text{BERT}_{\text{large}}$	335M	70.6	44.2	**72.8**	70.4	53.5	**78.3**	74.9	63.0	**75.1**	**68.3**	64.4	**68.3**
$\text{RoBERTa}_{\text{large}}$	355M	89.0	46.3	**89.3**	86.6	58.8	**89.5**	**75.6**	56.9	73.4	63.5	**64.4**	63.5
$\text{GLM}_{\text{xlarge}}$	2B	91.8	82.7	**91.9**	**90.3**	85.6	**90.3**	**74.1**	71.0	72.0	**95.2**	87.5	92.3
$\text{GLM}_{\text{xxlarge}}$	10B	**94.4**	87.8	92.5	**93.1**	89.9	**93.1**	**75.7**	71.8	74.0	**95.2**	94.2	93.3

Results on SuperGLUE development set. P-tuning v2 surpasses P-tuning & Lester et al. (2021) on models smaller than 10B, matching the performance of fine-tuning across different model scales. (FT: fine-tuning; PT: Lester et al. (2021) & P-tuning; PT-2: P-tuning v2; **bold**: the best; underline: the second best).

论文还通过消融实验研究了不同任务上Prompt Length的影响：

- 针对简单任务：如情感分析，较短的Prompt（~20）即可取得不错的效果。
- 针对复杂任务：如阅读理解，需要更长的Prompt（~100）。



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

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)

总之，P-Tuning v2是一种在**不同规模和任务中都可与微调相媲美的提示方法。**P-Tuning v2对从330M到10B的模型显示出一致的改进，并在序列标注等困难的序列任务上以很大的幅度超过了Prompt Tuning和P-Tuning。P-Tuning v2可以成为微调的综合替代方案和未来工作的基线（Baseline）。

6.五大方法详细对比表

方法	可训练参数位置	是否改变输入序列	是否需要每层干预	是否需要 VERBALIZER	支持序列标注（如 NER）	典型可训练参数占比	适用任务	是否使用 MLP/LSTM
Prompt Tuning	输入 embedding 层（软 prompt tokens）	是（序列变长）	仅输入层	是（依赖 label words）	否	~0.01%	NLU（分类）	否
P-Tuning (v1)	输入 embedding 层 + LSTM/MLP	是（混合模板）	仅输入层	是	否	~0.01%	NLU（few-shot 分类）	是
Prefix Tuning	每层 Key/Value 中的 prefix 向量	否	是	是（用 LM head）	否	~0.1%	NLG（生成）	原始用 MLP
P-Tuning v2	每层 Key/Value 中的 prefix + 分类头	否	是	**否**（用分类头）	**是**	~0.1%–3%	**NLU + NLG + 序列标注**	否（直接优化）
BitFit	所有 bias 参数（Attention/MLP/LN）+ 分类头	否	（但 bias 遍布各层）					