Large Language Model

大语言模型训练——LoRA篇

钱炳州 朱奕杰 赵洛右

Fine-tuning（微调）

- 在预训练模型的基础上进一步调整模型
- 优势
  - 站在巨人的肩膀上
  - 训练成本低
  - 适用于小数据集

Fine-tuning（微调）

- 预训练模型规模不断增大，对硬件要求高
- 目标设计变得繁琐复杂
- 无法直接适配下游任务
- 优化成本提高
- ……

Prompt微调

- 通过模板将下游任务转换为适合预训练模型处理的形式
- 模板设计：

输入x: I love this movie. -> X: I love this movie. Overall, it was a _____ movie.

标签y: good

- 答案搜索：在答案空间中进行搜索，找出得分最高的值填充到对应空槽中
- 答案映射：将填充的值对应到最终的输出标签 y（good）

硬提示/离散提示（Hard Prompt / Discrete Prompt）

- 人为设计提示词
- 特点：
  - 人类认为不错的离散提示对于语言模型来说不一定好
  - 离散提示的选取对于预训练模型的影响非常大

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

Table 1. Case study on LAMA-TREx P17 with bert-base-cased. A single-word change in prompts could yield a drastic difference.

软提示/连续提示（Soft Prompt / Continuous Prompt）

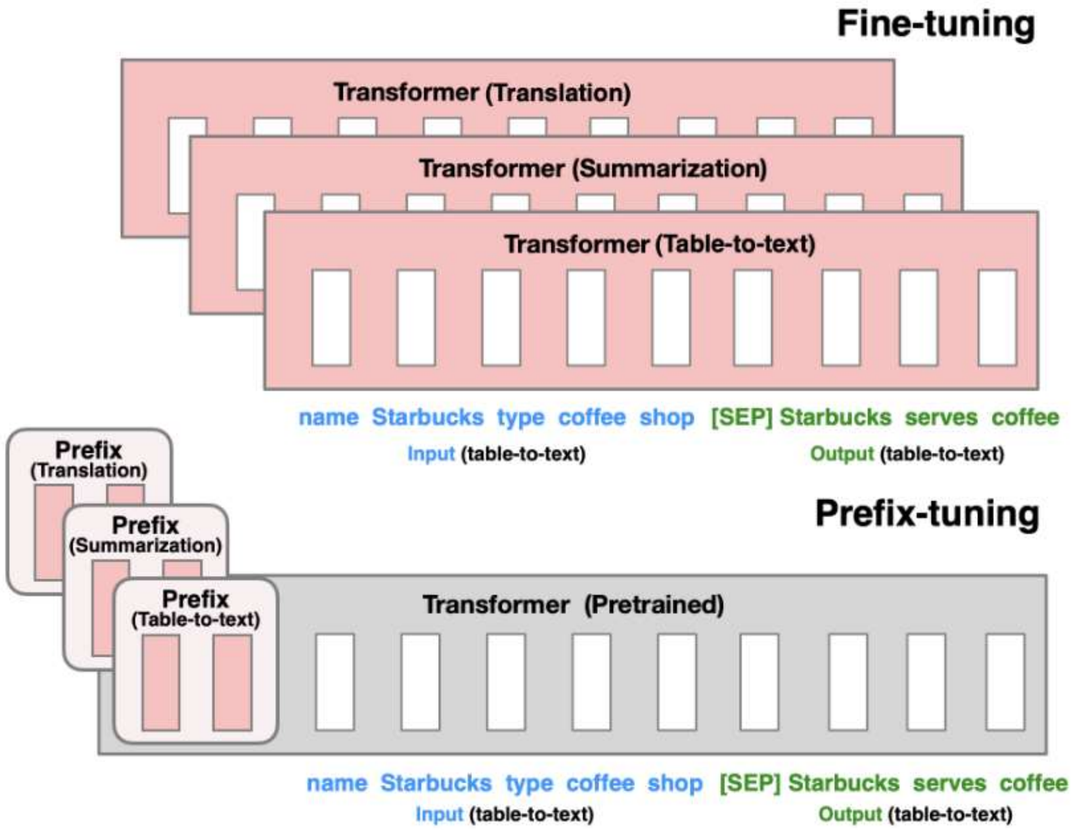
- 将Prompt的生成本身作为任务
- 分类
  - Prefix-tuning
  - P-tuning
  - Prompt-tuning

Prefix Tuning

- 一种用于生成式任务的技术，它在保留模型参数的同时，通过调整每个编码器层和输入层前插入的前缀激活函数来进行微调。

基本概念

- 在生成式任务中，模型应能根据输入生成输出，例如生成文章、翻译句子等。
- 通常情况下，我们会为每个任务设计一组适当的前缀，这些前缀可以帮助模型更好地理解任务的要求。每个任务都有其独特的前缀，不同任务之间不共享前缀。
- 在使用时，挑选任务相关的Prefix和Transformer进行组装，可实现可插拔式的应用



举例说明

假设我们的生成式任务是将英文句子翻译成法文。在传统的方法中，我们将英文句子直接输入给模型，然后模型根据预训练参数生成相应的翻译结果。

但是，在Prefix Tuning中，我们会在英文句子前插入一个特定的前缀。这个前缀可以是任务相关的，例如："Translate English to French: "。通过这个前缀，模型能够更好地理解输入的任务要求，产生更准确的翻译结果。

- 输入： "I love cats."
- 前缀： "Translate English to French: "
- 输入："Translate English to French: I love cats."

实际操作

- 在一句话前面加上若干个连续的“伪”token，这些伪token不必是词表中真实的词，而只是若干个可调的自由参数。
- 对于transformer的**每一层**都在真实的句子表征前面插入若干个可训练的"virtual token" embedding。
- 可以通过反向传播错误来调整每个编码器层和输入层前插入的前缀激活函数。每个任务都有其独特的前缀参数，而这些参数可以根据任务的需要进行调整。

如果另一个生成式任务是将英文翻译成德文，前缀则与前一个任务不同："Translate English to German: "。通过微调过程，模型可以根据不同的前缀参数来适应不同的翻译任务，从而产生准确的翻译结果。

P-Tuning

- P-tuning是一项针对NLU（自然语言理解）任务的工作，它提出了一种针对BERT类双向语言模型和单向语言模型的模板化方法。该方法的目标是通过对输入进行模板调整，以提高模型在下游任务上的性能。

工作原理

- 对双向语言模型模板：(P1, x, P2, [MASK], P3)
- 对单向语言模型模板：(P1, x, P2, [MASK])。模型可以在输入文本的末尾位置进行填充或替换。

P1表示前缀1；x表示输入文本；P2表示前缀2；[MASK]表示特殊的掩码标记；P3表示后缀

举例说明

- 输入："这个电影很[MASK]"。
- 模版化后输入：很[MASK]我觉得这个电影非常[prompt tokens]，真的很[prompt tokens]

前缀1（P1）为"很[MASK]"，即在输入文本的前部插入了一个固定的前缀。

输入文本（x）为"我觉得这个电影非常"，保留了原始的输入文本。

前缀2（P2）为"[prompt tokens]"，表示与任务相关的提示标记。

最后的后缀（P3）为"真的很[prompt tokens]"，在输入文本的后部插入了一个固定的后缀。

举例说明

- 输入："今天的天气很[MASK]"。
- 模版化后输入："今天的天气很[MASK][prompt tokens]，出门的话就要记得[prompt tokens]"。

前缀1（p1）和输入文本（x）都是”今天的天气很[MASK]

前缀2（P2）为"出门的话就要记得[prompt tokens]"，表示与任务相关的提示标记。

改进

1. 为了处理相关联的提示标记，作者使用了LSTM来对提示进行编码，以捕捉它们之间的关系。

2. 在输入上添加了锚点（anchor），例如在RTE（Recognizing Textual Entailment）任务中，可以在输入文本中加入一个问号作为锚点，这样有助于模型更好地理解任务的要求，进而提高性能。

之前的Prompt模型主要在小样本上表现出色，而P-tuning终于在整个数据集上超越了精调方法，提升了模型的性能。

Prompt Tuning

- 一种用于下游任务的技术，它通过调整输入文本中的提示信息（prompt）来改进预训练模型的性能。
- 不需要修改中间层或添加额外的输出层，它仅仅在输入的提示部分进行调整

Span Corruption

这种方法是基于T5预训练模型的改进版本。T5是一种序列到序列模型，用于处理自然语言处理任务。在"Span Corruption"中，我们使用T5来预测输入文本中的句子空缺内容，并使用标记来表示空缺位置。

假设输入文本是

"Thank you 〈X〉 me to your party 〈Y〉 week"。

我们希望模型能够填充〈X〉和〈Y〉这两个空缺位置。那么在训练过程中，我们将输入作为模型的输入，而输出则是填充空缺的内容。

模型输出："〈X〉 for inviting 〈Y〉 last 〈Z〉"。

〈Z〉标记表示输出的结尾。

Span Corruption + Sentinel

在这种方法中，为了使微调过程更接近预训练的状态，我们在所有下游任务的头部插入一个特殊的哨兵标记（sentinel）。这个哨兵标记可以看作是一个额外的提示，它帮助模型更好地理解下游任务的输入。

下游任务是情感分类，给定一个句子来判断其情感是积极还是消极。在句子的开头插入哨兵标记。

- 输入："情感是：〈S〉这个餐厅真是太好了！"。

通过标记，模型能够通过哨兵标记更好地理解任务的要求，并对情感进行正确分类

LM Adaptation

- 一种延续了T5模型的自监督训练方法，但以"LM"（Language Modeling）为目标
- 模型进行少量额外的自监督训练，以便更好地适应生成式任务。在这个训练过程中，模型的目标是基于给定的上下文预测下一个可能出现的token
- 通过LM Adaptation，作者希望将模型转换成一个更类似于GPT-3的模型。

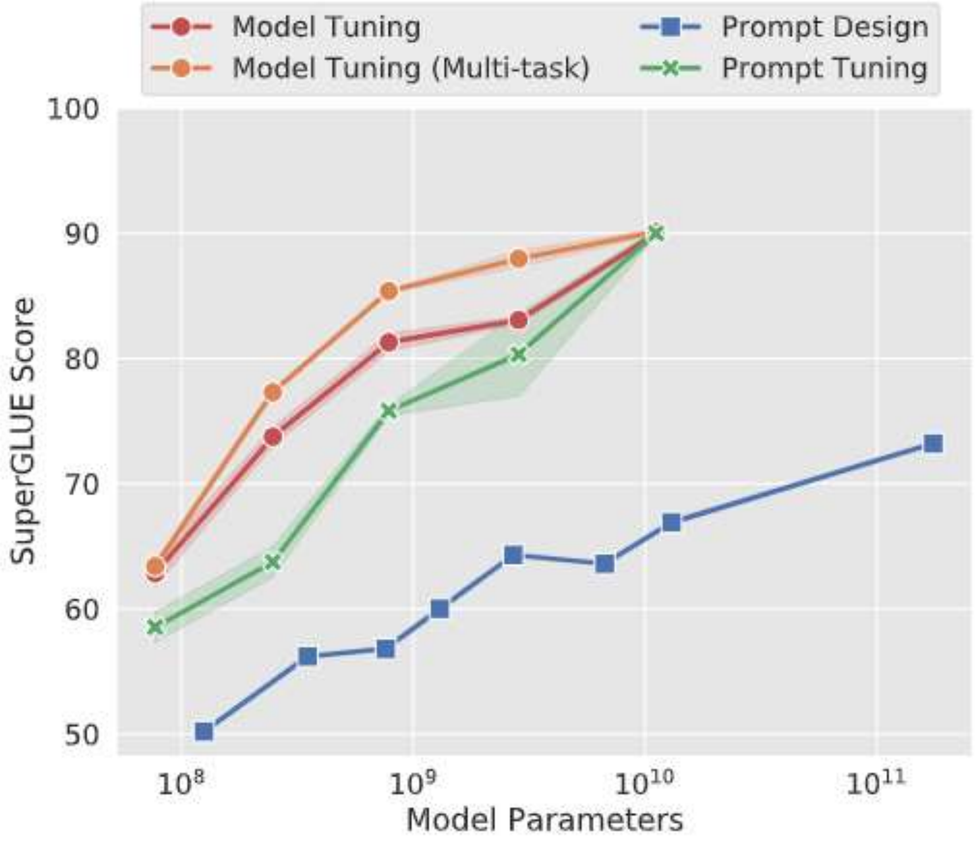
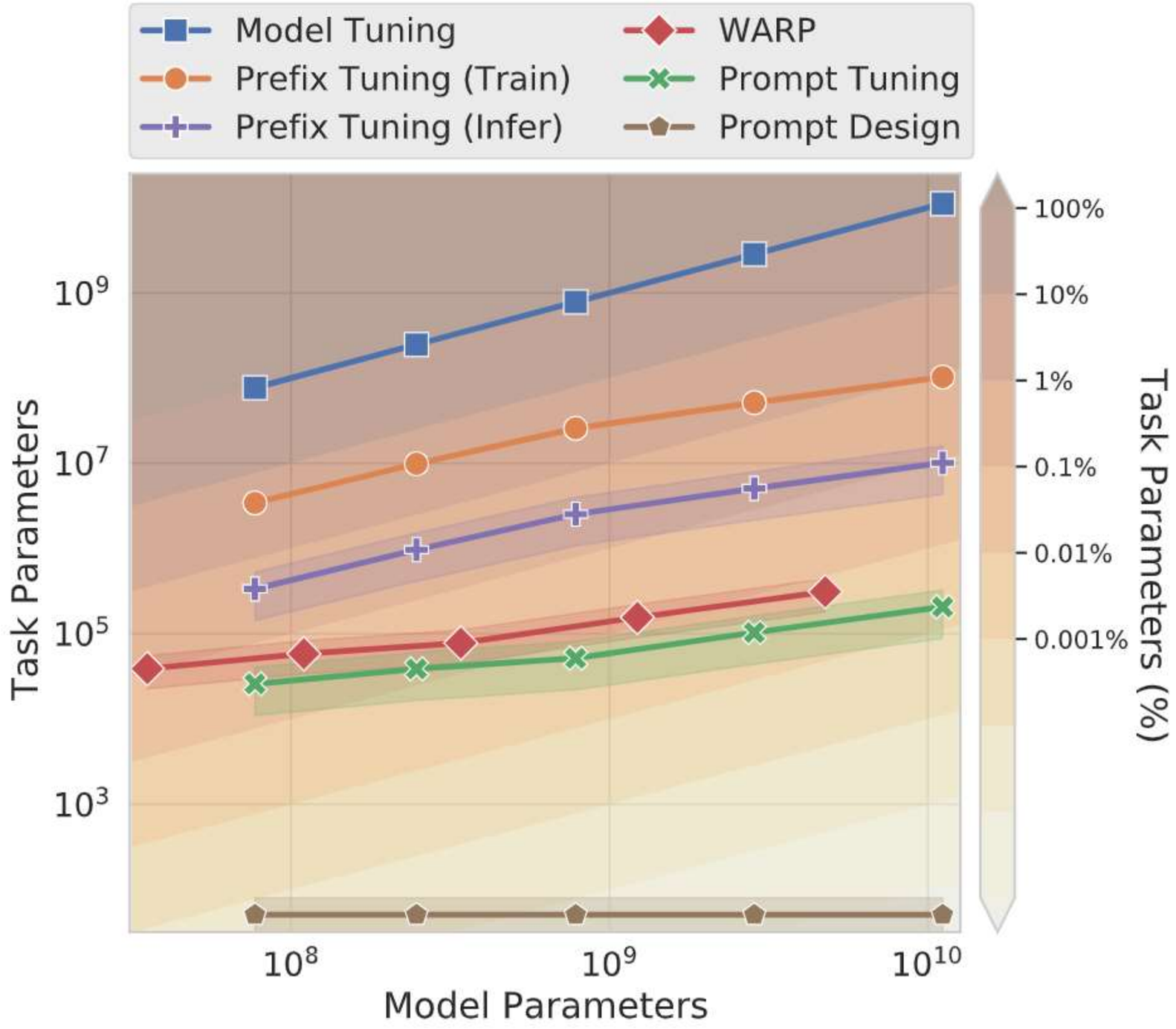


Figure 1: Standard **model tuning** of T5 achieves strong performance, but requires storing separate copies of the model for each end task. Our **prompt tuning** of T5 matches the quality of model tuning as size increases, while enabling the reuse of a single frozen model for all tasks. Our approach significantly outperforms few-shot **prompt design** using GPT-3. We show mean and standard deviation across 3 runs for tuning methods.


Model Tuning
Prefix Tuning (Train)
Prefix Tuning (Infer)
WARP
Prompt Tuning
Prompt Design
Task Parameters
10^9
10^7
10^5
10^3
10^8
10^9
10^{10}
Model Parameters
100%
10%
1%
0.1%
0.01%
0.001%
Task Parameters (%)

LoRa（Low-Rank Adaptation）

现有方法缺点

- classic fine-tune：完全更新模型参数

  缺点：参数量过大，效率低下，存储所有微调后的模型相当困难

- Adapter：在模型网络中插入Adapter层，在不影响原参数的情况下，对Adapter层内参数进行更新

  缺点：设计引入了串行步骤，造成推理延迟“显著”提高（3%~30%，序列越短越明显），需要在效率和模型质量之间权衡

- prompt tuning

  缺点：优化难度大，其性能在可训练参数增加时非单调变化。更重要的是，固定一部分序列长度必然会导致下游任务可用的序列长度减少，可能导致模型性能不如其他方法。

LoRA

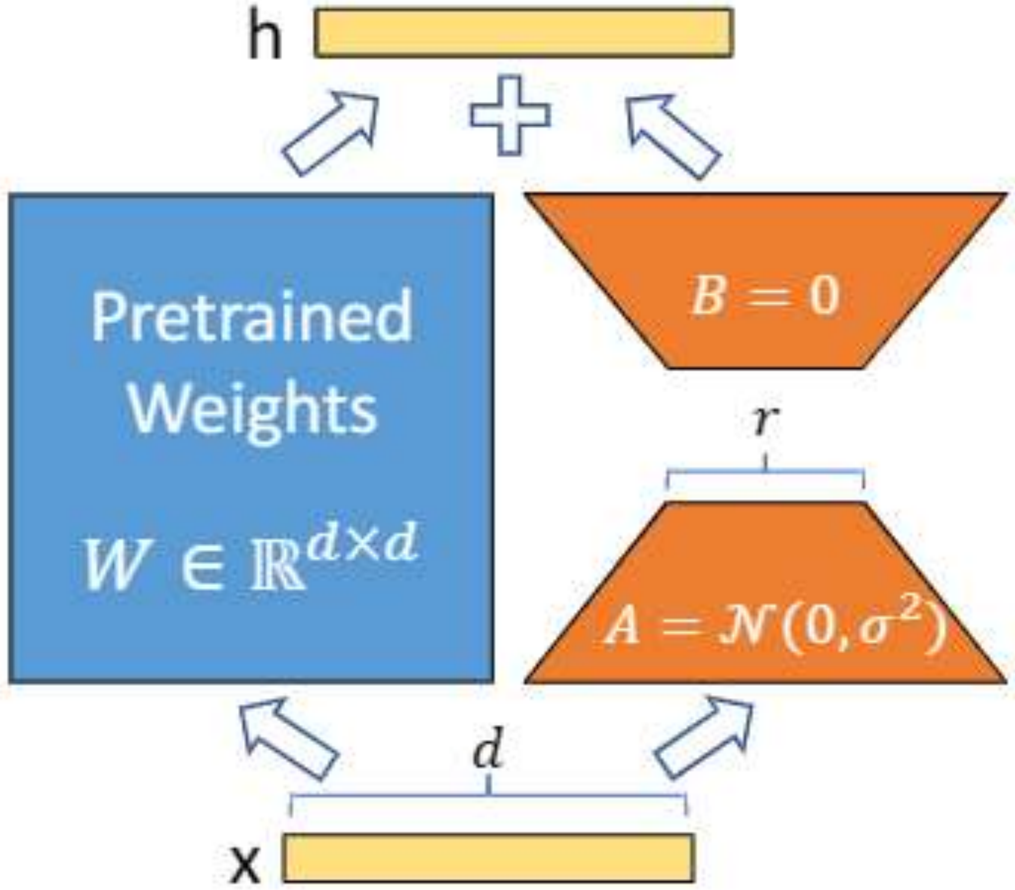
特点

- 不改变预训练模型权重，而在模型外添加旁路，用训练旁路中的矩阵模拟微调过程。

优点

- 不引入额外延迟，只需要将旁路参数与模型参数合并，用新参数推理结果即可。

假设：模型权重的更新矩阵有一个低秩



本征维度

Intrinsic Dimensionality

定义：目标函数达到精确解决优化问题所需的最小维度

在2020年，一项由Aghajanyan等人进行的研究衡量了模型微调的本征维度，尝试寻找对每个任务进行微调时需要多少空余参数才能大概解决优化问题

他们提出——预训练本身是在为下游NLP任务最小化本征维度，**预训练实际上是一个学习压缩平均NLP任务的框架**

本征维度计算

计算目标函数的精确本征维度非常困难，但可以使用启发式方法计算上界。

假设 $\theta^D = [\theta_0, \theta_1, ..., \theta_m]$ 是模型 $f(\cdot, \theta)$ 的D个参数。相比直接优化模型参数 θ^D 的损失函数，我们只关注参数的子空间，通过参数化d（d < D）个维度对模型进行微调：

$$\theta^D = \theta_0^D + P(\theta^d)$$

其中 $P : \mathbb{R}^d \to \mathbb{R}^D$，将参数从低维d映射到高维D。微调过程中，只有 θ^d 是变量，其他都是定值。

实现方法

定义达到训练完整权重矩阵效果的90%为令人满意的效果（如果完整模型训练后准确率85%，那么调整后目标准确率为0.9*85%=76.5%），称这个最小维度为 d_{90}。

- **DID.** 随机从所有参数中挑选 θ^d，称这种方法找的的本征维度为Direct Intrinsic Dimension

$$\theta^D = \theta_0^D + \theta^d M,\ \ M = HG\Pi HB$$

- **SAID.** 对模型的每一层参数单独进行映射，以参数之和作为本征维度Structure Aware Intrinsic Dimension

$$\theta_i^D = \theta_{0,i}^D + \lambda_i P(\theta^{d-m})_i$$

本征维度计算结果

- MRPC：包含约3700个训练样本的段落文本语义一致性预测任务
- QQP：包含约363k个训练样本的问题文本语义一致性预测任务

	SAID		DID	
Model	MRPC	QQP	MRPC	QQP
BERT-Base	1608	8030	1861	9295
BERT-Large	1037	1200	2493	1389
RoBERTa-Base	896	896	1000	1389
RoBERTa-Large	**207**	**774**	322	**774**

Table 1: Estimated d_{90} intrinsic dimension for a set of sentence prediction tasks and common pre-trained models. We present both the *SAID* and *DID* methods.

结论

- 预训练模型效果越好，本征维度越小
- 训练集规模越大，本征维度越大

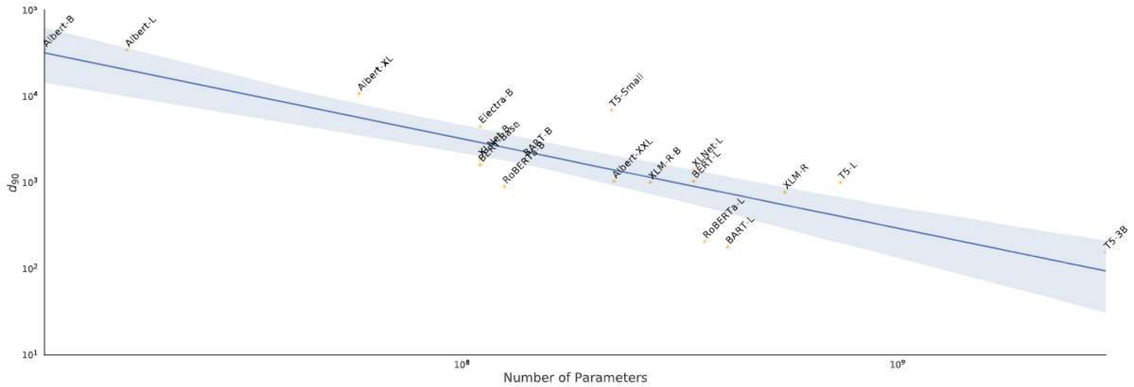


Figure 3: We calculate the intrinsic dimension for a large set of pre-trained models using the SAID method on the MRPC dataset.

重新理解LoRA

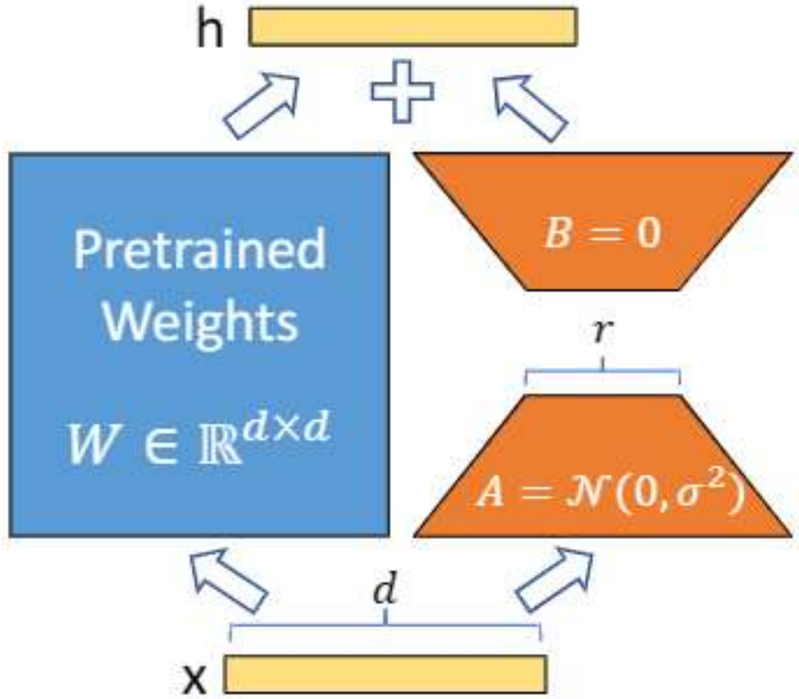
现有预训练权重矩阵 $W_0 \in \mathbb{R}^{d\times k}$ ，将权重更新表示为

$$W_0 + \triangle W = W_0 + BA$$

其中 $B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times k}$，秩 $r \ll min(d, k)$（可任意取值）。

在训练中，预训练权重矩阵保持不变，不接受梯度更新，而将A, B作为训练参数矩阵。

使用高斯随机函数初始化A，使用0初始化B，因此$\triangle W$在训练开始时为0。



实现细节

在Transformer结构包含两个模块——自注意力、MLP（多层感知器）：

- 自注意力模块有4个权重矩阵 W_q, W_k, W_v, W_o
- MLP模块有2个权重矩阵

实验中，LoRA只微调 W_q, W_k, W_v 中的一个，相当于为下游任务调整时只变更注意力权重，而不变化MLP模块。

对于添加的LoRA矩阵，取秩r=4。

LoRA的优势

- 参数规模可以自由调节（修改秩r）
- 可以分级进行微调（attach 不同结构的权重）
- 保留了模型的多任务能力
- 可解释性强

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, and $\text{DeBERTa}_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around $\pm 0.5\%$, MNLI-m around $\pm 0.1\%$, and SAMSum around $\pm 0.2/\pm 0.2/\pm 0.1$ for the three metrics.

Chinese-LLaMA-Alpaca

项目地址：https://github.com/ymcui/Chinese-LLaMA-Alpaca

LLaMA：2023年Facebook Meta发布的最新Transformer decoder大语言模型，主要基于英文数据进行训练，采用了包括**预归一化、SwiGLU 激活函数、旋转嵌入**等一系列优化措施。

Chinese-LLaMA-Alpaca

- 增加2000中文词汇，增强LLaMA的中文理解能力
- 采用LoRA进行模型高效训练和部署

Pre-train Stage-1

Chinese-LLaMA没有完全训练一个新模型，而是以原LLaMA得到权重作为初始状态，在中文数据集上继续进行预训练。

中文数据集，仅有LLaMA训练语料大小的0.5%

方法

- 固定Transformer encoder中的参数——最小对化模型的扰动
- 训练embedding层参数——适应新添加的中文词向量

Pre-train Stage-2

方法

- 使用LoRA，并行调整Transformer注意力模块权重
- 训练embedding层和LM head层

LM head层：出现在大部分类GPT结构中，作为模型输出层，将hidden_states张量的最后一个维度映射到词典维度

Instruction Fine-tuning

由于微调采用的是 Stanford Alpaca 提出的对LLaMA的自指导微调方法来训练指令服从模型，此阶段产生的模型称为Chinese Alpaca。

方法

- 使用ChatGPT基于少量的指令数据Prompt，迭代进行指令数据的生成
- 使用LoRA，并行调整Transformer MLP模块权重

Reference

参考文献

- Liu X, Zheng Y, Du Z, et al. GPT understands, too[J]. arXiv preprint arXiv:2103.10385, 2021.
- Liu X, Ji K, Fu Y, et al. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks[J]. arXiv preprint arXiv:2110.07602, 2021.
- Li X L, Liang P. Prefix-tuning: Optimizing continuous prompts for generation[J]. arXiv preprint arXiv:2101.00190, 2021.
- Lester B, Al-Rfou R, Constant N. The power of scale for parameter-efficient prompt tuning[J]. arXiv preprint arXiv:2104.08691, 2021.
- Aghajanyan A, Zettlemoyer L, Gupta S. Intrinsic dimensionality explains the effectiveness of language model fine-tuning[J]. arXiv preprint arXiv:2012.13255, 2020.
- Hu E J, Shen Y, Wallis P, et al. Lora: Low-rank adaptation of large language models[J]. arXiv preprint arXiv:2106.09685, 2021.
- Touvron H, Lavril T, Izacard G, et al. Llama: Open and efficient foundation language models[J]. arXiv preprint arXiv:2302.13971, 2023.
- Wang Y, Kordi Y, Mishra S, et al. Self-Instruct: Aligning Language Model with Self Generated Instructions[J]. arXiv preprint arXiv:2212.10560, 2022.
- Cui Y, Yang Z, Yao X. Efficient and Effective Text Encoding for Chinese LLaMA and Alpaca[J]. arXiv preprint arXiv:2304.08177, 2023.

谢谢大家