博学谷
www.boxuegu.com
黑马程序员线上品牌

Prompt-Tuning方法进阶

一样的教育，不一样的品质

目录

Contents

01

超大规模参数模型Prompt-Tuning方法

超大规模参数模型Prompt-Tuning方法

- 近两年来，随着Prompt-Tuning技术的发展，对于超过10亿参数量的模型来说，Prompt-Tuning所带来的增益远远高于标准的Fine-tuning. 如GPT-3模型，只需要设计合适的模板或指令即可以实现免参数训练的零样本学习.
- 根本原因：模型参数量足够大，训练过程中使用了 足够多的语料，同时设计的 预训练任务足够有效.

面向超大规模的模型的Prompt-Tuning方法：

01

上下文学习 In-Context Learning

直接挑选少量的样本作为该任务的提示.

02

指令学习 Instruction-Tuning

构建任务指令集，促使模型根据任务指令做出反馈

03

思维链 Chain-of-Thought

给予或激发模型具有推理和解释的信息，通过线性链式的模式指导模型生成合理的结果.

02

上下文学习方法的应用

In-Context Learning

- In-Context learning（ICL）最早在GPT3中提出，旨在从训练集中挑选少量的标注样本，设计任务相关的指令形成提示模板，用于指导测试样本生成相应的结果.

zero-shot learning	给出任务的描述，然后提供测试数据对其进行预测，直接让预训练好的模型去进行任务测试.
one-shot learning	给出任务的描述，在进行新数据预测前，插入一个样本做指导，相当于给一个例子让模型理解，然后再提供测试数据对其进行预测.
few-shot learning	给出任务的描述，在进行新数据预测前，插入N个样本做指导. 相当于给N个例子让模型理解，然后再提供测试数据对其进行预测.

03

指令学习方法的应用

Instruction-Tuning

- 其实Prompt-Tuning本质上是对下游任务的指令，简单的来说：就是告诉模型需要做什么任务，输出什么内容. 上文我们提及到的离散或连续的模板，本质上就是一种对任务的提示.
- 因此，在对大规模模型进行微调时，可以为各种类型的任务定义指令，并进行训练，来提高模型对不同任务的泛化能力.

Prompt VS Instruction：

1：带女朋友去了一家餐厅，她吃的很开心，这家餐厅太__了！

=

2：判断这句话的情感：带女朋友去了一家餐厅，她吃的很开心。选项：A=好，B=一般，C=差

Prompt为第一种模式，Instruction为第二种. 很明显：做判别比做生成更容易.

Instruction-Tuning

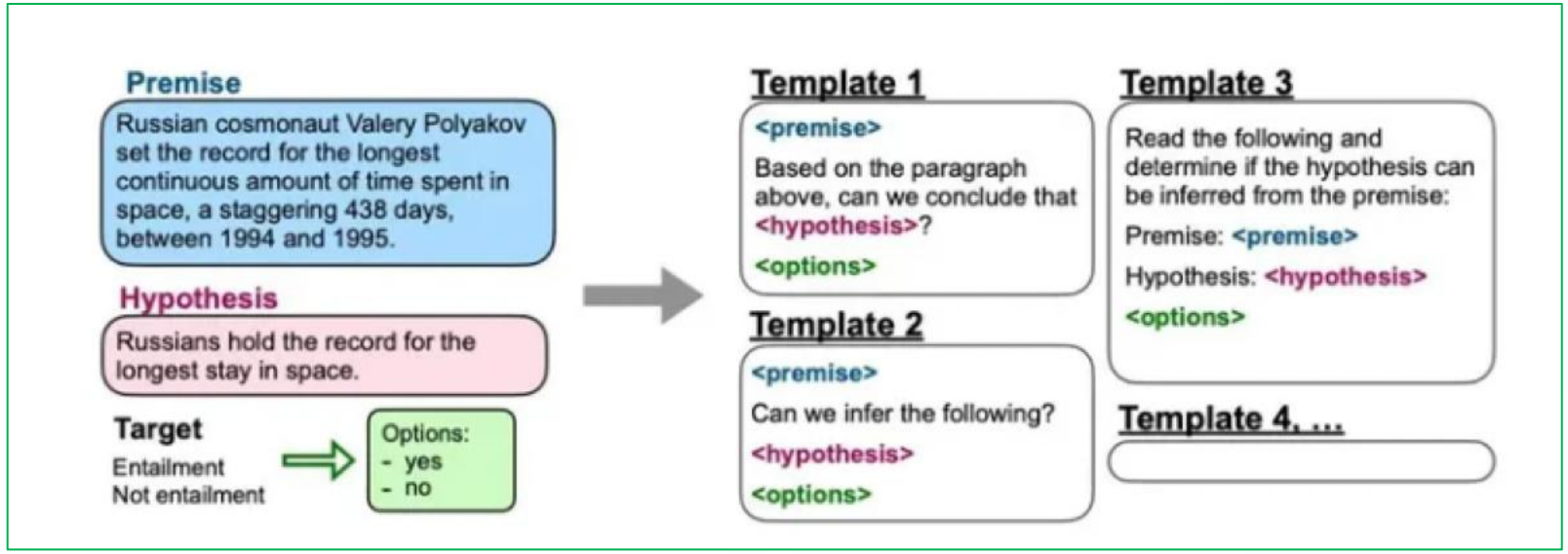
Instruction-Tuning和Prompt-Tuning的核心一样，就是去发掘语言模型本身具备的知识.

Instruct-Tuning的形式（以电影评论二分类举例）：

在对电影评论进行二分类的时候，最简单的提示模板(Prompt)是“. It was [mask].”，但是其并没有突出该任务的具体特性，我们可以为其设计一个能够突出该任务特性的模板(加上Instruction)，例如“The movie review is . It was [mask].”，然后根据mask位置的输出结果通过Verbalizer映射到具体的标签上。这一类具备任务特性的模板可以称之为指令Instruction.

Instruction-Tuning

如何实现Instruction-Tuning?



为每个任务设计10个指令模版，测试时看平均表现.

Instruction-Tuning

指令学习和提示学习的不同点：

01

Prompt是去激发语言模型的补全能力，比如给出上半句生成下半句、或者做完形填空

02

Instruction-Tuning则是激发语言模型的理解能力，通过给出更明显的指令/指示，让模型去理解并做出正确的action.

03

Promp-Tuningt在没有精调的模型上也能有一定效果，但是Instruct-Tuning则必须对模型精调，让模型知道这种指令模式.

思考总结

Thinking summary

1.什么是指令学习？

答案：通过给出更明显的指令/指示，让模型去理解并做出正确的action.

2. 指令学习和Prompt的区别？

答案：指令学习激发语言模型的理解能力；指令学习激发语言模型的补全能力.

04

思维链方法的实现

Chain-of-Thought

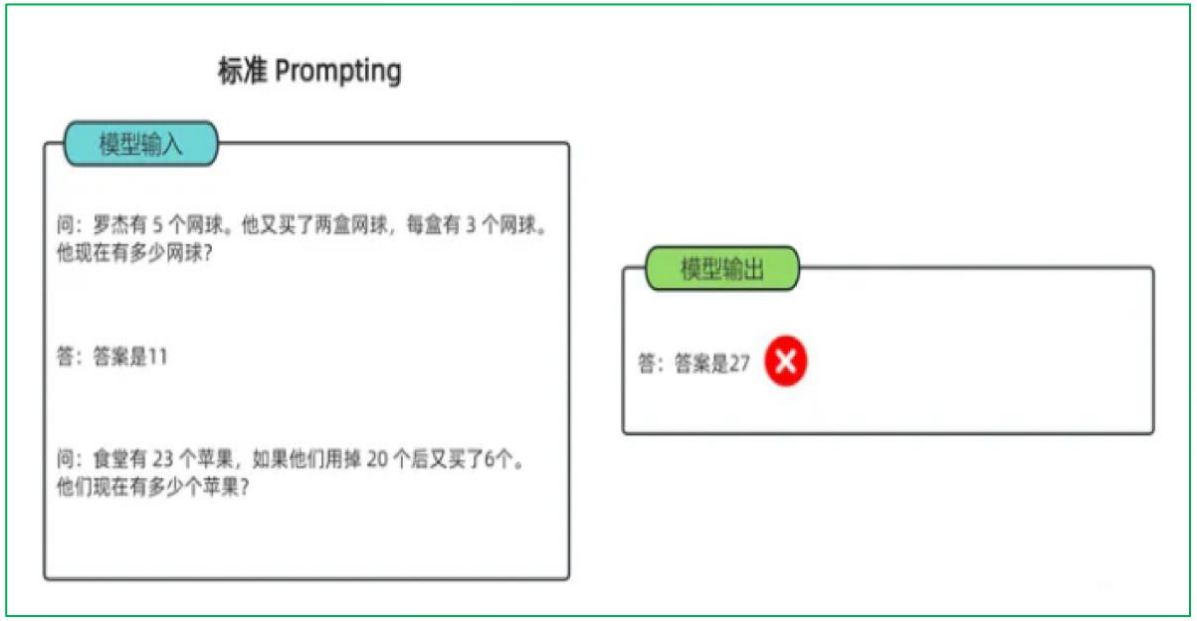
- 思维链 (Chain-of-thought，CoT) 的概念是在 Google 的论文 "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models" 中被首次提出.

- 思维链（CoT）是一种改进的提示策略，用于提高 LLM 在复杂推理任务中的性能，如算术推理、常识推理和符号推理.

- 思维链是一种离散式提示学习，更具体地，大模型下的上下文学习（即不进行训练，将例子添加到当前样本输入的前面，让模型一次输入这些文本进行输出完成任务），相比于之前传统的上下文学习（即通过$x_1, y_1, x_2, y_2, ... x_{test}$作为输入来让大模型补全输出ytest），思维链多了中间的推导提示.

Chain-of-Thought理解

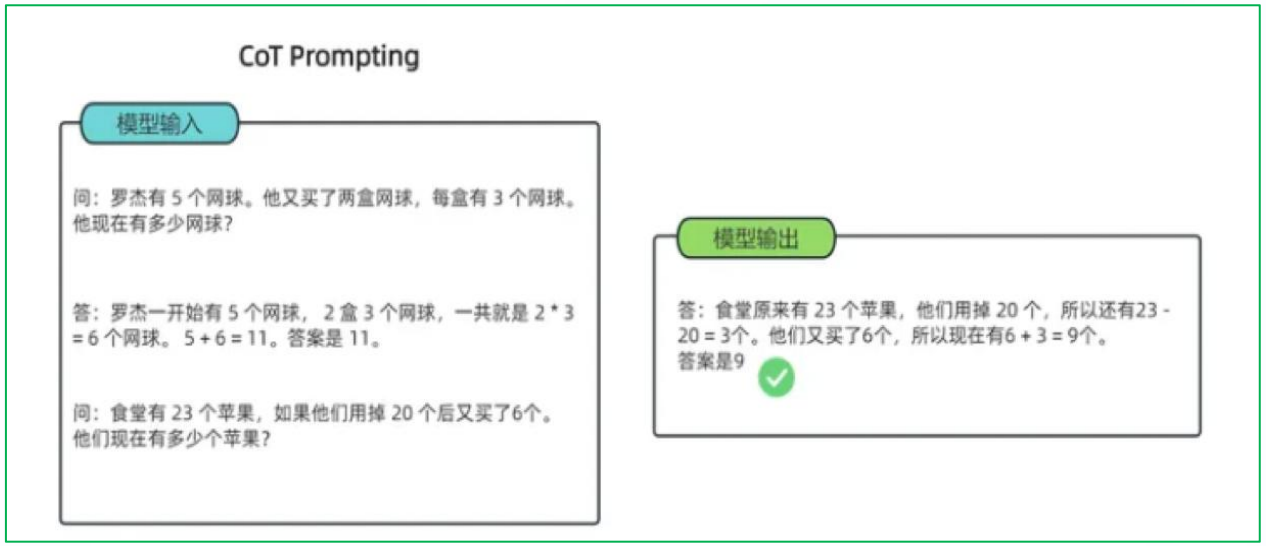
- 求解一个数学题为例，理解cof思想：



可以看到模型无法做出正确的回答。但如果说，我们给模型一些关于解题的思路，就像我们数学考试，都会把解题过程写出来再最终得出答案，不然无法得分.

Chain-of-Thought理解

- 求解一个数学题为例，理解cof思想：



可以看到，类似的算术题，思维链提示会在给出答案之前，还会自动给出推理步骤：

“罗杰先有5个球，2盒3个网球等于6个，5 + 6 = 11”

“食堂原来有23个苹果，用了20个，23-20=3；又买了6个苹果，3+6=9.

Chain-of-Thought分类

Few-shot CoT

ICL 的一种特殊情况，它通过融合 CoT 推理步骤，将每个演示〈input，output〉扩充为〈input,CoT,output〉

Zero-shot CoT

直接生成推理步骤，然后使用生成的 CoT 来导出答案.（其中 LLM 首先由 “Let’s think step by step” 提示生成推理步骤，然后由 “Therefore, the answer is” 提示得出最终答案。他们发现，当模型规模超过一定规模时，这种策略会大大提高性能，但对小规模模型无效，显示出显著的涌现能力模式）

Chain-of-Thought特点

逻辑性	思维链中的每个思考步骤都应该是有逻辑关系的，它们应该相互连接，从而形成一个完整的思考过程.
全面性	思维链应该尽可能地全面和细致地考虑问题，以确保不会忽略任何可能的因素和影响.
可行性	思维链中的每个思考步骤都应该是可行的，也就是说，它们应该可以被实际操作和实施.
可验证性	思维链中的每个思考步骤都应该是可以验证的，也就是说，它们应该可以通过实际的数据和事实来验证其正确性和有效性

思考总结

Thinking summary

1.什么是思维链方法？

答案：相比于之前传统的上下文学习（即通过$x_1, y_1, x_2, y_2, \ldots . x_{test}$作为输入来让大模型补全输出ytest），思维链多了中间的推导提示.

2. 思维链的分类？

答案：Few-shot CoT以及Zero-shot CoT

05

PEFT大模型参数高效微调方法原理

PEFT介绍

PEFT（Parameter-Efficient Fine-Tuning）参数高效微调方法是目前大模型在工业界应用的主流方式之一，PEFT 方法仅微调少量或额外的模型参数，固定大部分预训练参数，大大降低了计算和存储成本，同时最先进的 PEFT 技术也能实现了与全量微调相当的性能.

PEFT的优势：

该方法可以使 PLM 高效适应各种下游应用任务，而无需微调预训练模型的所有参数，且让大模型在消费级硬件上进行全量微调（Full Fine-Tuning）变得可行.

PEFT分类

Prefix/Prompt-Tuning

在模型的输入或隐层添加k个额外可训练的前缀 伪tokens，只训练这些前缀参数

Adapter-Tuning

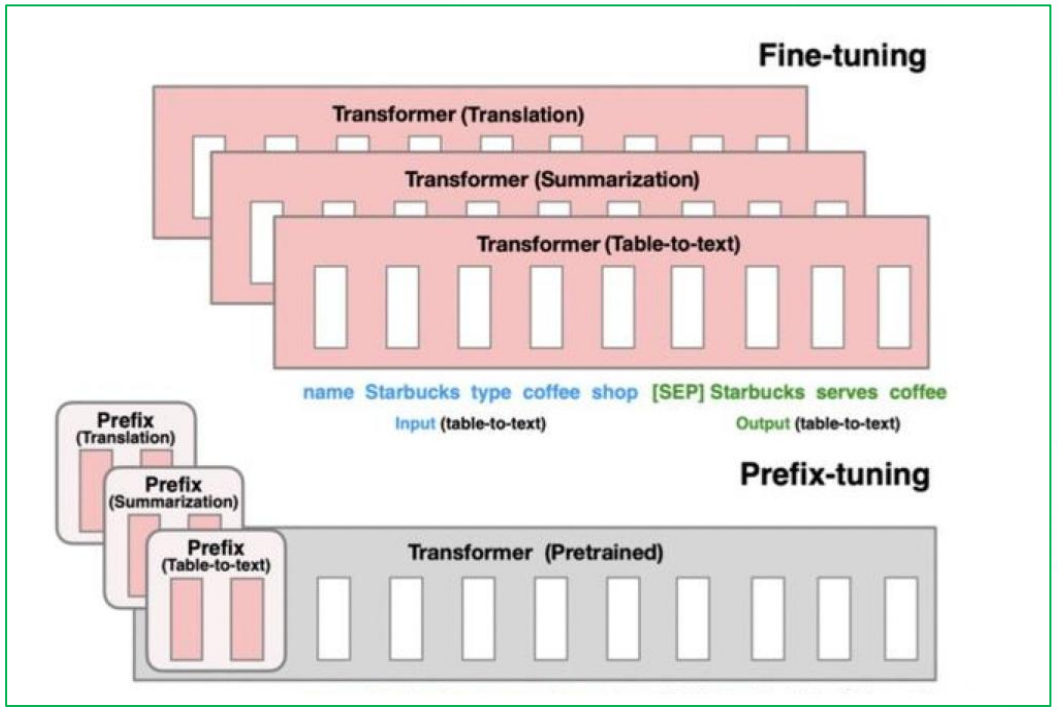
将较小的神经网络层或模块插入预训练模型的每一层，这些新插入的神经模块称为 adapter（适配器），下游任务微调时也只训练这些适配器参数.

LoRA

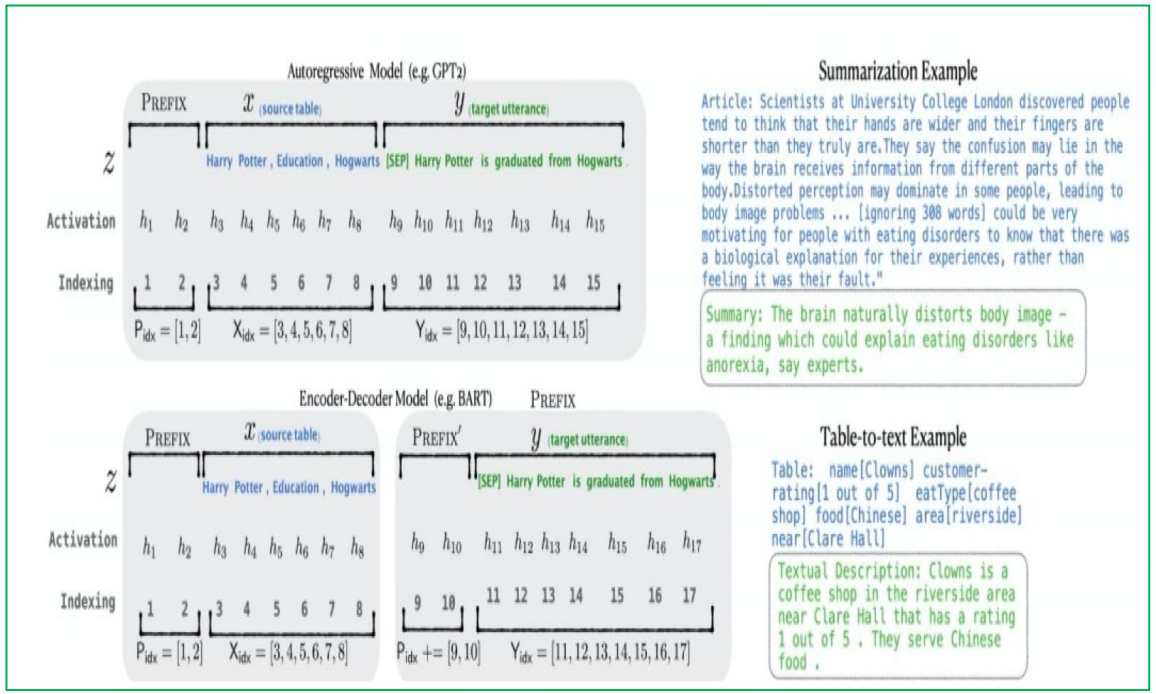
通过学习小参数的低秩矩阵来近似模型权重矩阵 W 的参数更新，训练时只优化低秩矩阵参数.

Prefix-Tuning

2021年论文《Prefix-Tuning: Optimizing Continuous Prompts for Generation》中提出了 Prefix Tuning 方法，该方法是在输入 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 Transformer 中的其他部分参数固定.



Prefix-Tuning任务形式



- Prefix-Tuning 在输入前添加前缀，即 z=[Prefix,x,y] ，P_{idx}为前缀序列的索引，$|P_{idx}|$为前缀的长度。前缀索引对应着由 θ 参数化的向量矩阵 P_θ，维度为$|P_{idx}| \times \dim(h_i)$.

- 注意：由于直接更新 Prefix 的参数会导致训练不稳定，作者在 Prefix 层前面加了 MLP 结构(相当于将Prefix分解为更小维度的 Input 与 MLP 的组合后输出的结果)，训练完成后，只保留 Prefix 的参数.

Prefix-Tuning特点

对比之前的微调方式：

对比P-Tuning

- Prefix-Tuning 是将额外的embedding加在开头，看起来更像模仿Instruction指令，而P-Tuning 位置不固定.
- Prefix-Tuning 通过在每个层都添加可训练参数，通过MLP初始化，而P-Tuning只在输入的时候加入embedding，并通过LSTM+MLP初始化.

对比Prompt-Tuning

- Prompt Tuning 方式可以看做是 Prefix Tuning 的简化，只在输入层加入 prompt tokens，并不需要加入 MLP 进行调整来解决难训练的问题.

思考总结

Thinking summary

1.什么是Prefix Tuning?

答案：该方法是在输入 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 Transformer 中的其他部分参数固定..

2. Prefix-Tuning和P-Tuing的区别?

答案：1.Prefix-Tuning 是将额外的embedding加在开头，而P-Tuning位置不固定；2.Prefix-Tuning 通过在每个层都添加可训练参数，通过MLP初始化，而P-Tuning只在输入的时候加入embedding，并通过LSTM+MLP初始化.

3. Prefix-Tuning和Prompt-Tuing的区别?

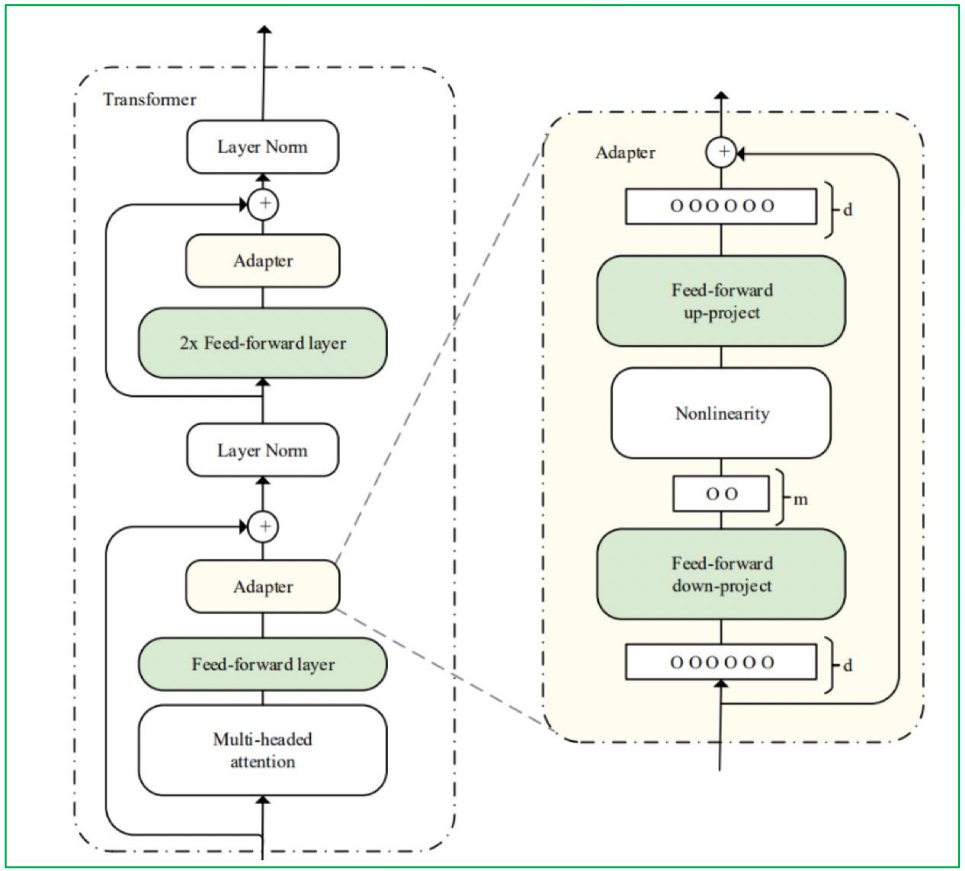
答案：Prompt Tuning 方式可以看做是 Prefix Tuning 的简化，只在输入层加入 prompt tokens，并不需要加入 MLP 进行调整来解决难训练的问题

Adapter Tuning

2019年谷歌的研究人员首次在论文《Parameter-Efficient Transfer Learning for NLP》提出针对 BERT 的 PEFT微调方式，拉开了 PEFT 研究的序幕.

不同于Prefix Tuning这类在输入前添加可训练 prompt参数，以少量参数适配下游任务，Adapter Tuning 则是在预训练模型内部的网络层之间添加新的网络层或模块来适配下游任务. 当模型训练时，固定住原来预训练模型的参数不变，只对新增的 Adapter 结构进行微调.

Adapter Tuning 模型结构



- 首先是一个 down-project 层将高维度特征映射到低维特征.
- 然后过一个非线形层之后，再用一个 up-project 结构将低维特征映射回原来的高维特征
- 同时也设计了 skip-connection 结构，确保了在最差的情况下能够退化为identity（类似残差结构）

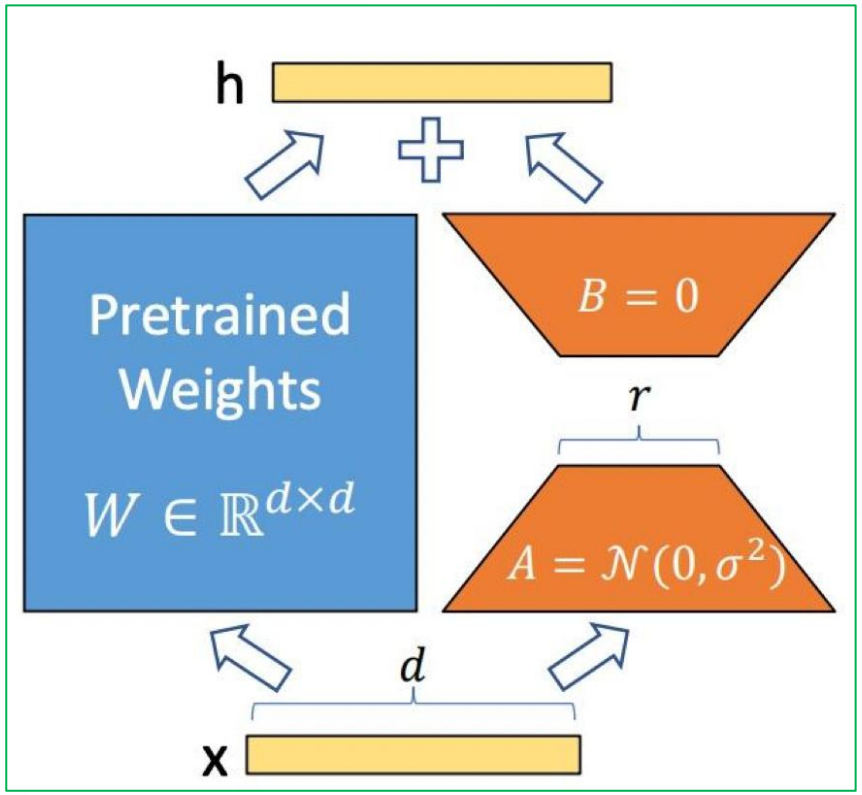
LoRA

低秩适应（Low-Rank Adaptation）是一种参数高效的微调技术，其核心思想是对大型模型的权重矩阵进行隐式的低秩转换，也就是：通过一个较低维度的表示来近似表示一个高维矩阵或数据集.

LoRA的产生：

上述Adapter Tuning 方法在 PLM 基础上添加适配器层会引入额外的计算，带来推理延迟问题；而 Prefix Tuning 方法难以优化，其性能随可训练参数规模非单调变化，更根本的是，为前缀保留部分序列长度必然会减少用于处理下游任务的序列长度. 因此微软推出了LoRA方法.

LoRA 原理



LoRA技术冻结预训练模型的权重，并在每个Transformer块中注入可训练层（称为秩分解矩阵），即在模型的Linear层的旁边增加一个“旁支”A和B。其中，A将数据从d维降到r维，这个r是LoRA的秩，是一个重要的超参数；B将数据从r维升到d维，B部分的参数初始为0。模型训练结束后，需要将A+B部分的参数与原大模型的参数合并在一起使用

LoRA 伪代码实现

```
input_dim = 768 # 例如，预训练模型的隐藏大小
output_dim = 768 # 例如，层的输出大小
rank = 8 # 低秩适应的等级'r'
W = ... # 来自预训练网络的权重，形状为 input_dim x output_dim
W_A = nn.Parameter(torch.empty(input_dim, rank)) # LoRA权重A
W_B = nn.Parameter(torch.empty(rank, output_dim)) # LoRA权重B初始化LoRA权重
nn.init.kaiming_uniform_(W_A, a=math.sqrt(5))
nn.init.zeros_(W_B)
def regular_forward_matmul(x, W):
  h = x @ W
  return h
def lora_forward_matmul(x, W, W_A, W_B):
  h = x @ W # 常规矩阵乘法
  h += x @ (W_A @ W_B) * alpha # 使用缩放的LoRA权重,alpha缩放因子
  return h
```

LoRA方法是目前最通用、同时也是效果最好的微调方法之一.

思考总结

Thinking summary

1.什么是Adapter Tuning?

答案：在预训练模型内部的网络层之间添加新的网络层或模块来适配下游任务.

2.什么是LoRA?

答案：一种特殊的adapter.一种高效参数微调方法；对大型模型的权重矩阵进行隐式的低秩转换.

3.LoRA原理?

答案：冻结预训练模型的权重，并在每个Transformer块中注入可训练层（称为秩分解矩阵），即在模型的Linear层的旁边增加一个“旁支”A和B。其中，A将数据从d维降到r维，这个r是LoRA的秩，是一个重要的超参数；B将数据从r维升到d维，B部分的参数初始为0。模型训练结束后，需要将A+B部分的参数与原大模型的参数合并在一起使用

博学谷
www.boxuegu.com
黑马程序员线上品牌

扫码关注博学谷微信公众号