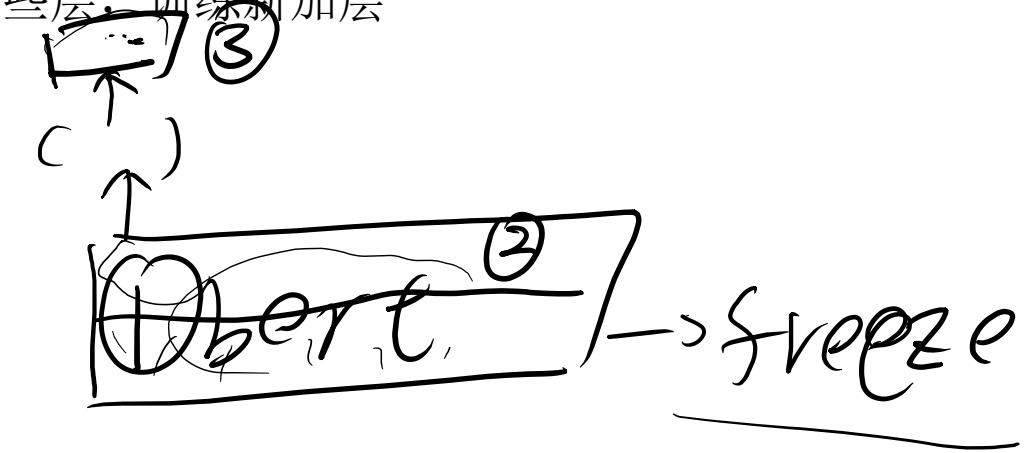
大模型微调

调参：
Batchsize=10
6*6b*10=360G 显存

Adam

Bert
T5
Bart

传统大模型微调：
1.微调所有参数
2.调整部分参数（冻结部分层）
3.对模型尾部加一些层，训练新加层



扫码了解更多

直接微调模型参数，使用场景：

1.数据量大 token数量>=可调参数

2.机器算力足 显存大小决定能否训练 显卡flops决定训练时间

3.场景非常垂直

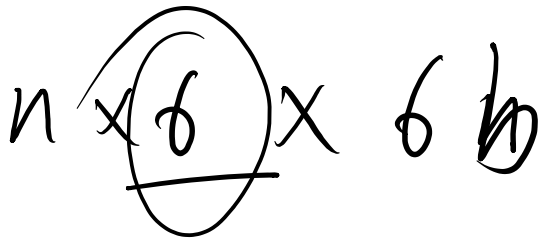
扫码了解更多

显存里面到底存了哪些数据：

- 模型：6b个float32 浮点数

Adam梯度下降法：

- 存前向结果
- 存之前步骤的梯度结果

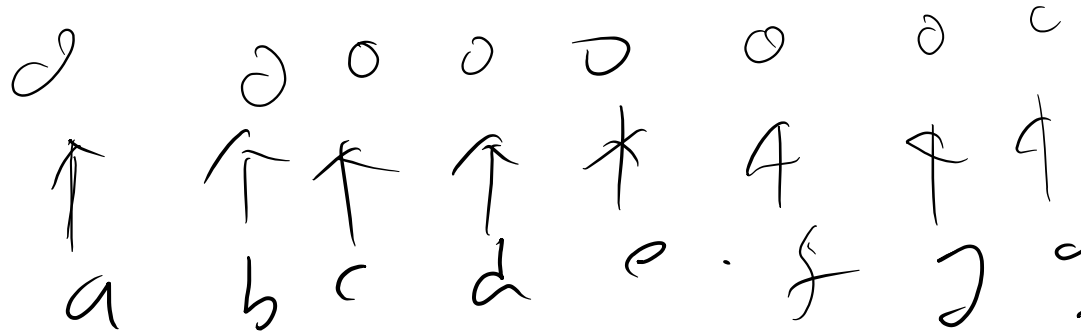


因为显存的限制：

模型可训练参数不能太多

Batchsize不能太大

Max_length 不能太大



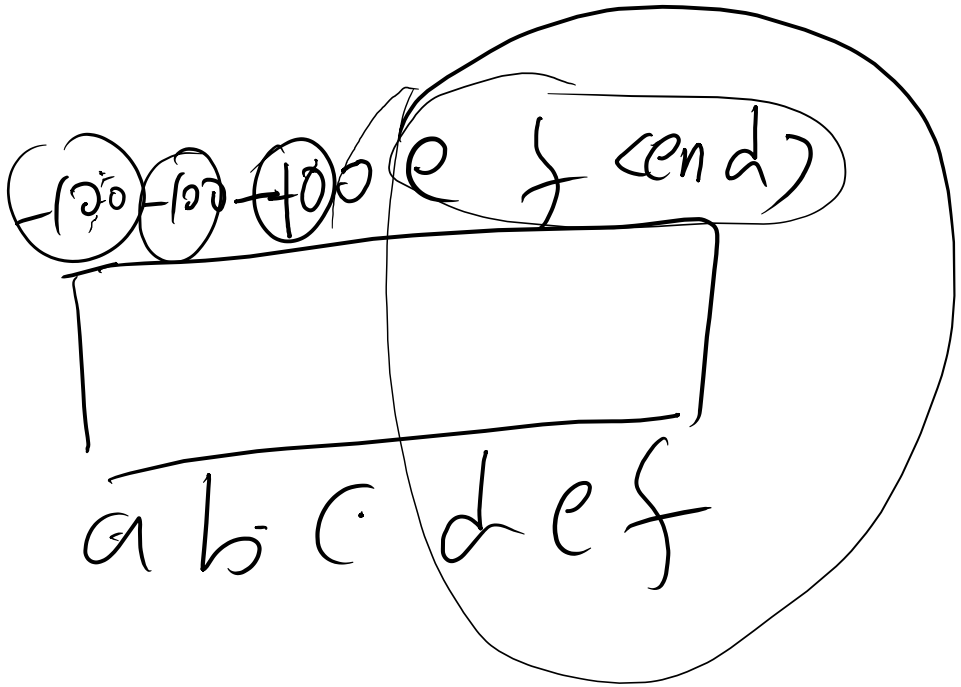
分享经验：
Maxlength：一般会统计平均长度（去除异常值） 平均长度*1.5
确定好训练的参数
Batchsize理论上来说，越大越好

扫码了解更多

bpe

Bpe分词：常见组合的字符串拼接在一起
Sentencepiece：Google出现分词工具
它可以实现bpe
把所有的文本都转成utf-8
所以有可能：
1。几个汉字对应一个token
2.2个token对应一个汉字

max length



扫码了解更多

模型本质是个语言模型
Gpt为例

Q A

Q: abcd

A: efgh

训练

扫码了解更多

大模型调参难点，为什么很少直接微调？
1.参数多，显存不容易放下
2.参数多，需要对应更大数据
3.参数多，不容易收敛
4.参数多，调参时间过长

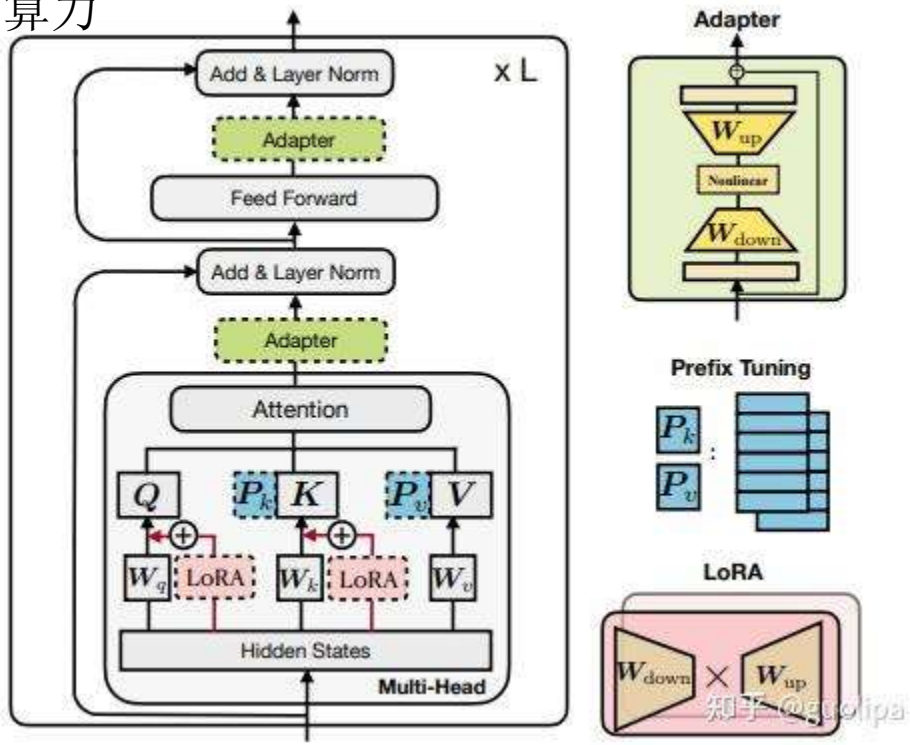
做大模型：
一半技术 一半运气

扫码了解更多

参数高效微调方法（Parameter-Efficient Fine-Tuning，PEFT）

- Prefix-Tuning / Prompt-Tuning：在模型的输入或隐层添加 k 个额外可训练的前缀 tokens（这些前缀是连续的伪 tokens，不对应真实的 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。

节约内存 节省算力



扫码了解更多

问题：今天能否打篮球

Prompt+问题：今天天晴，温度25度。今天能否打篮球

Prefix+问题：a b c d e今天能否打篮球 答案：今天可以打篮球

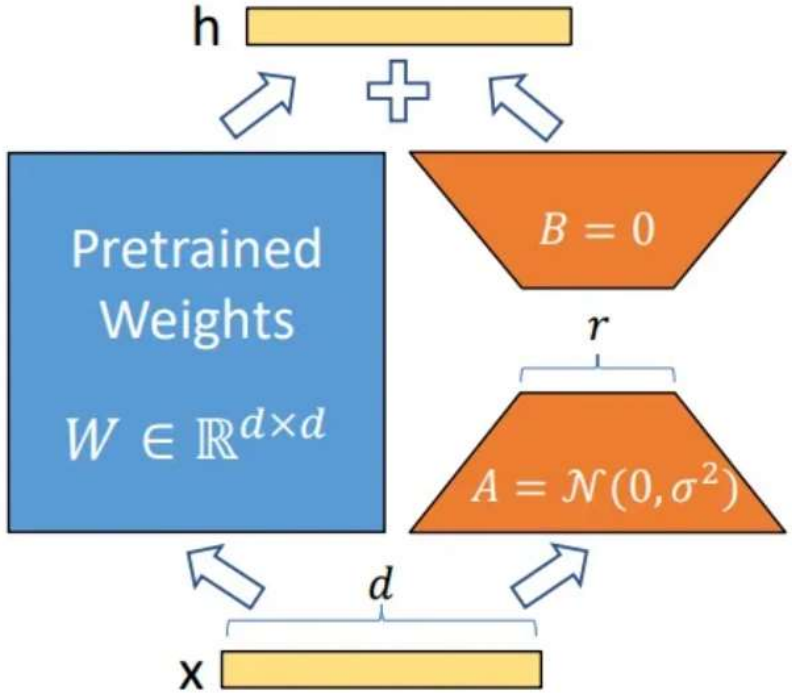
扫码了解更多

- Adapter Tuning 增加了模型层数，引入了额外的推理延迟
- Prefix-Tuning 难于训练，且预留给 Prompt 的序列挤占了下游任务的输入序列空间，影响模型性能
- P-tuning v2 很容易导致旧知识遗忘，微调之后的模型，在之前的问题上表现明显变差

扫码了解更多

大模型中的LoRA

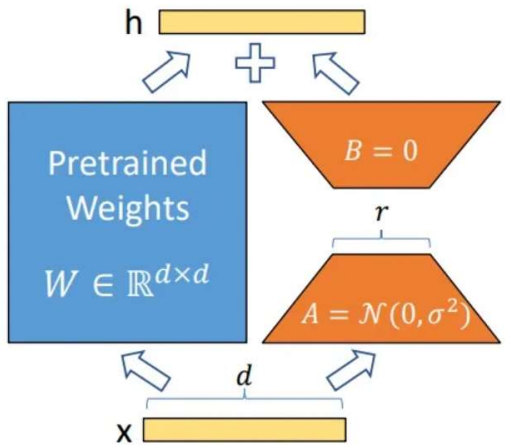
$$h = W_0 x + \Delta W x = W_0 x + BAx$$

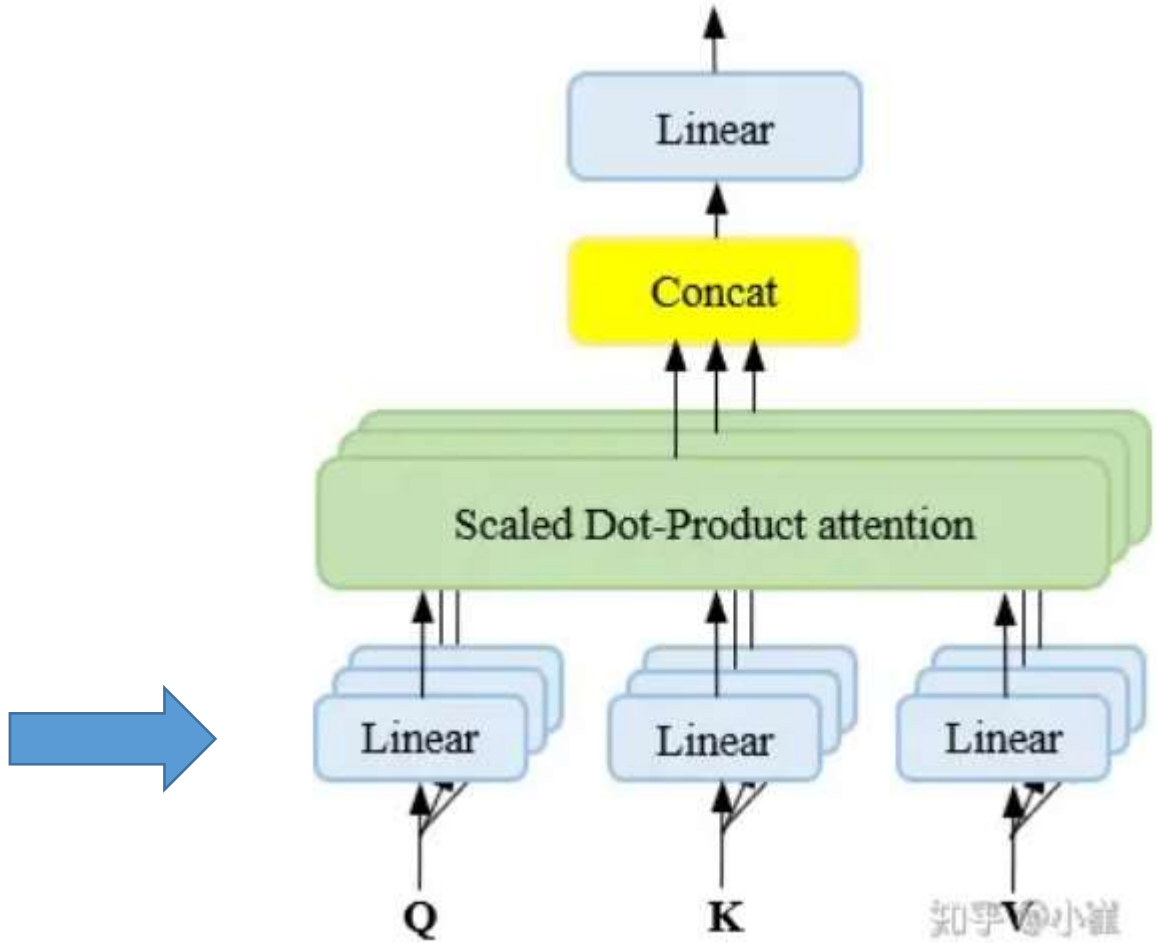


好处：节约内存

扫码了解更多

大模型中的LoRA





知乎@小崔

扫码了解更多

大模型中的LoRA

$$W = W_0 + UV, \qquad U \in \mathbb{R}^{m\times r}, V \in \mathbb{R}^{r\times n}$$

$$\frac{\partial \mathcal{L}}{\partial U} = \frac{\partial \mathcal{L}}{\partial W} V^\top, \quad \frac{\partial \mathcal{L}}{\partial V} = U^\top \frac{\partial \mathcal{L}}{\partial W}$$

$$U_{t+1} = U_t - \eta \frac{\partial \mathcal{L}}{\partial W_t} V_t^\top, \quad V_{t+1} = V_t - \eta U_t^\top \frac{\partial \mathcal{L}}{\partial W_t}$$

$$W_{t+1} = W_0 + U_{t+1}V_{t+1} = W_t + (U_{t+1}V_{t+1} - U_tV_t)$$

扫码了解更多

大模型中的LoRA

计算量并没有减少，但速度确更快，原因

1、只更新了部分参数：比如LoRA原论文就选择只更新Self Attention的参数，实际使用时我们还可以选择只更新部分层的参数；
2、减少了通信时间：由于更新的参数量变少了，所以（尤其是多卡训练时）要传输的数据量也变少了，从而减少了传输时间；
3、采用了各种低精度加速技术，如FP16、FP8或者INT8量化等。

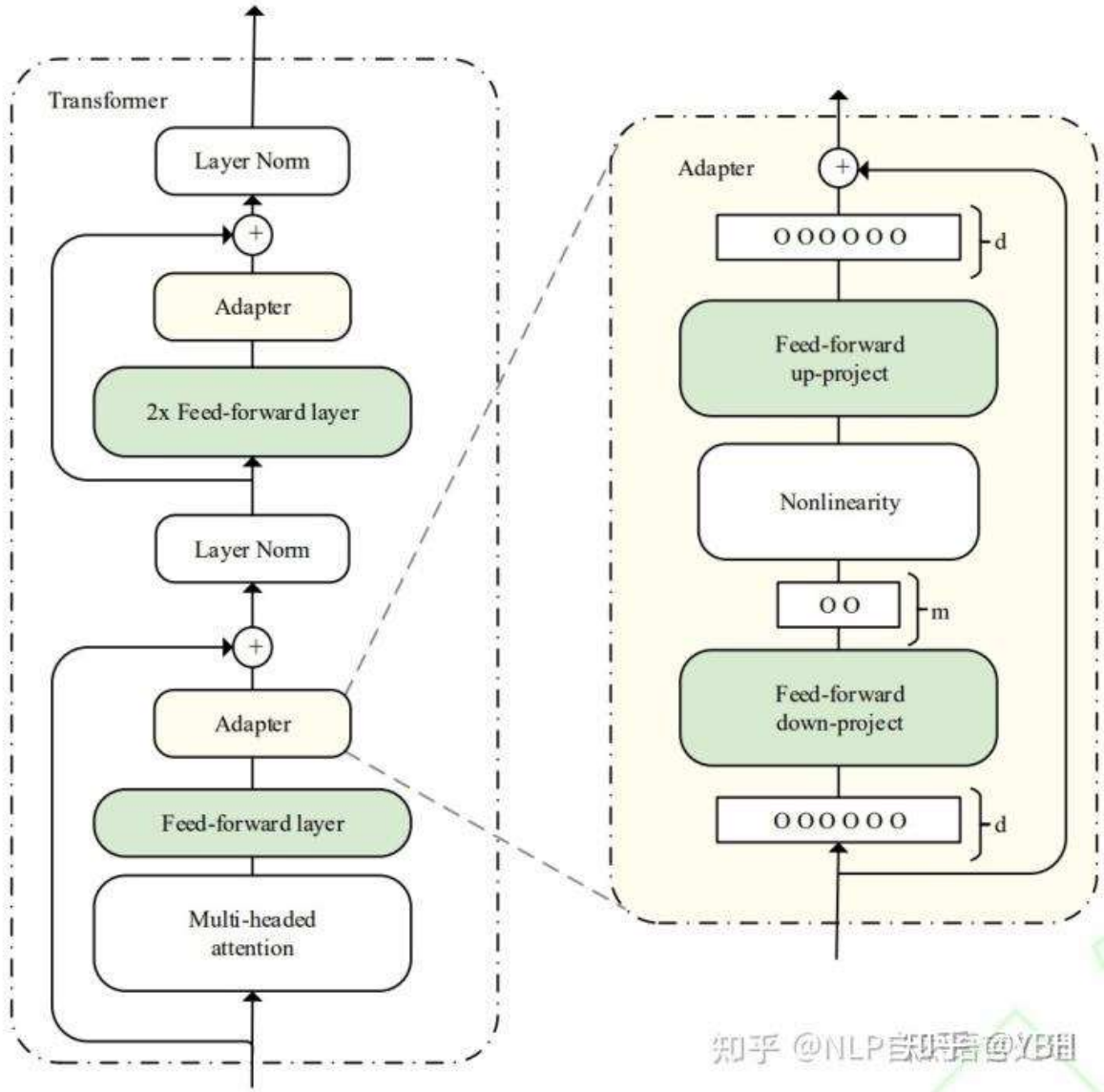
扫码了解更多

大模型中的LoRA

Rank确定方法

度量模型和垂直领域的差异性

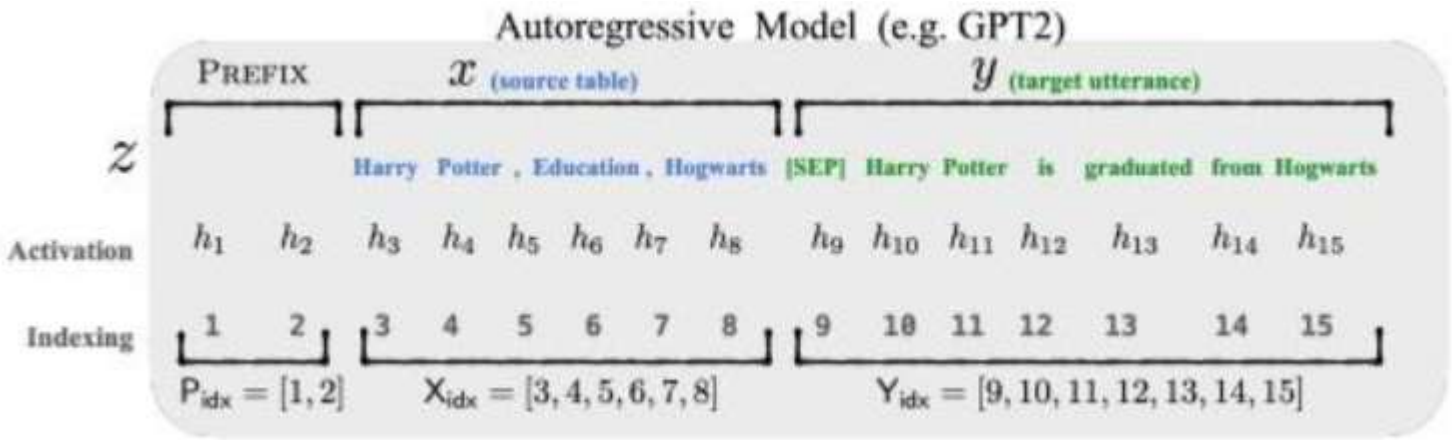
大模型中的Adapter



知乎 @NLP

扫码了解更多

大模型中的Prefix Tuning



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

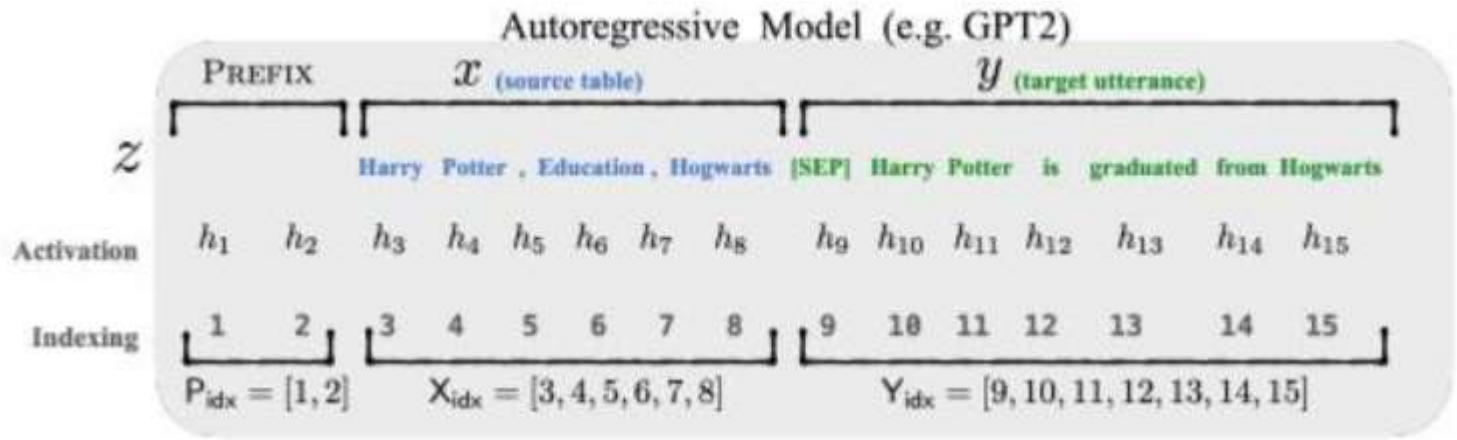
知乎 @xiaolipa

[Full vs Embedding-only]：Embedding-only 方法只在 embedding 层添加前缀向量并优化，而 Full 代表的 Prefix-tuning 不仅优化 embedding 层添加前缀参数，还在模型所有层的激活添加前缀并优化。实验得到一个不同方法的**表达能力增强链条：discrete prompting < embedding-only < prefix-tuning**。同时，Prefix-tuning 可以直接修改模型更深层的表示，避免了跨越网络深度的长计算路径问题。

[Prefix-tuning vs Infix-tuning]：通过将可训练的参数放置在 x 和 y 的中间来研究可训练参数位置对性能的影响，即 $[x; Infix; y]$，这种方式成为 Infix-tuning。实验表明 **Prefix-tuning 性能好于 Infix-tuning**，因为 prefix 能够同时影响 x 和 y 的隐层激活，而 infix 只能够影响 y 的隐层激活。

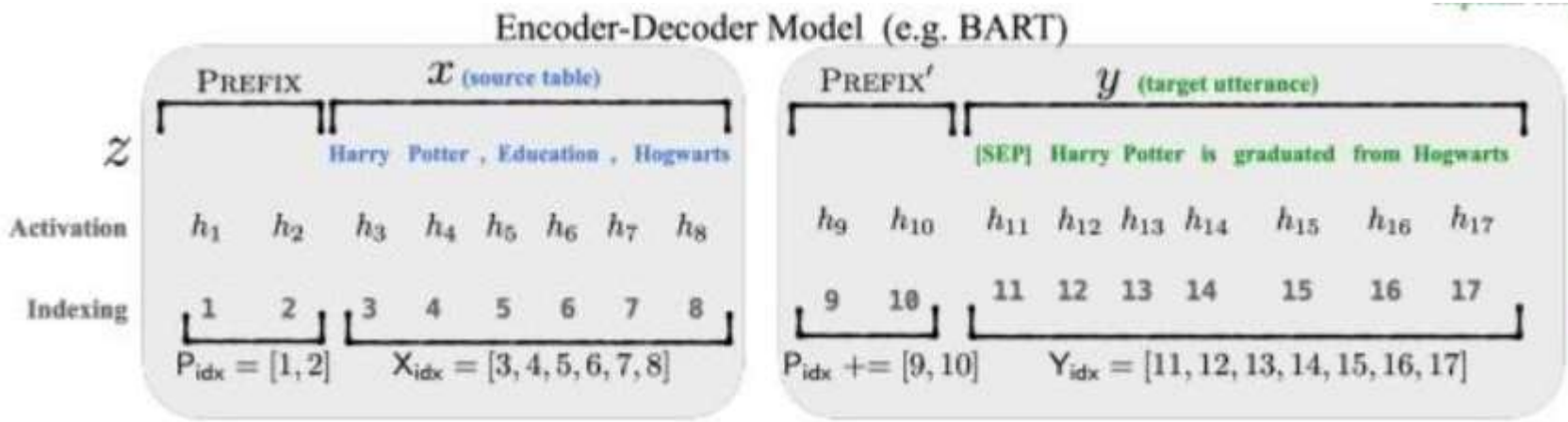
扫码了解更多

大模型中的Prefix Tuning



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

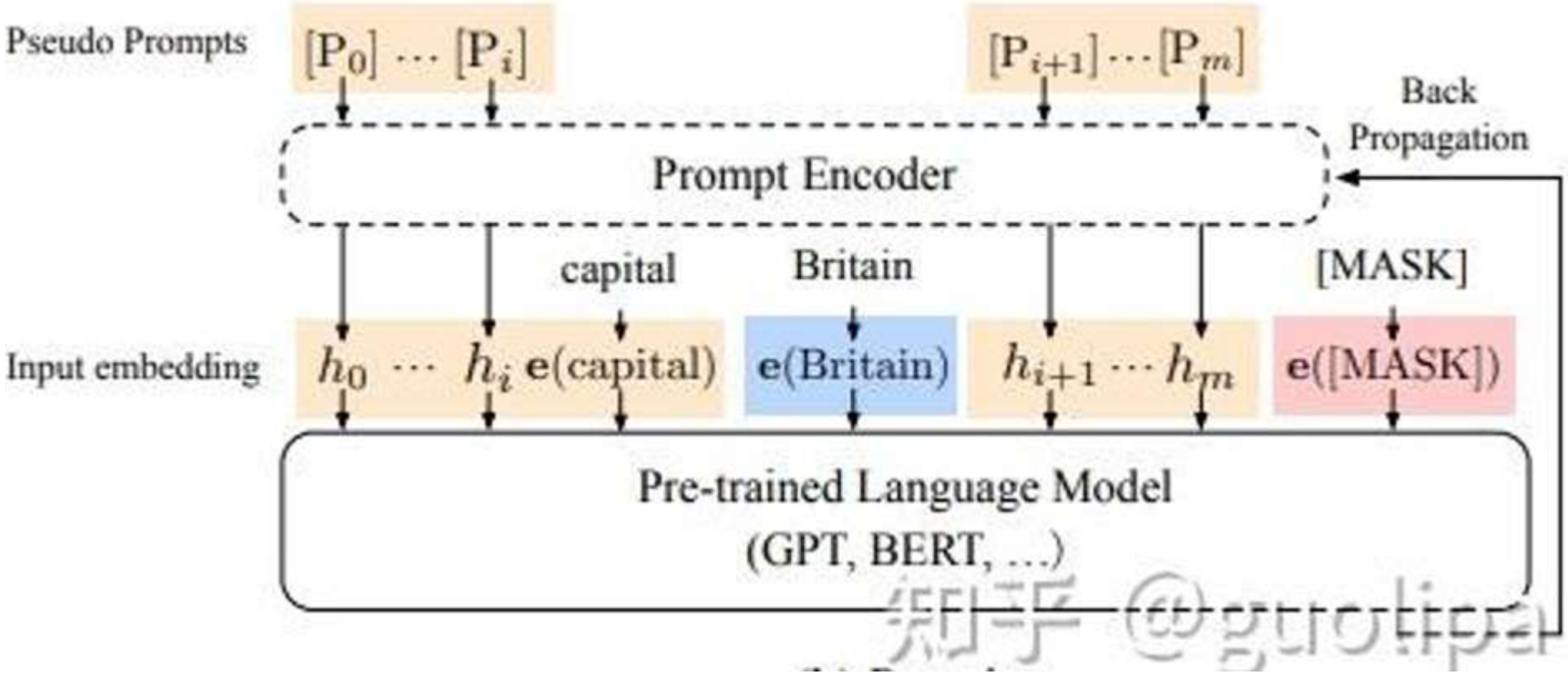
MLP_θ 对 P_θ 进行重参数化: $P_\theta[i,:] = MLP_\theta(P'_\theta[i,:])$



扫码了解更多

大模型中的P-Tuning

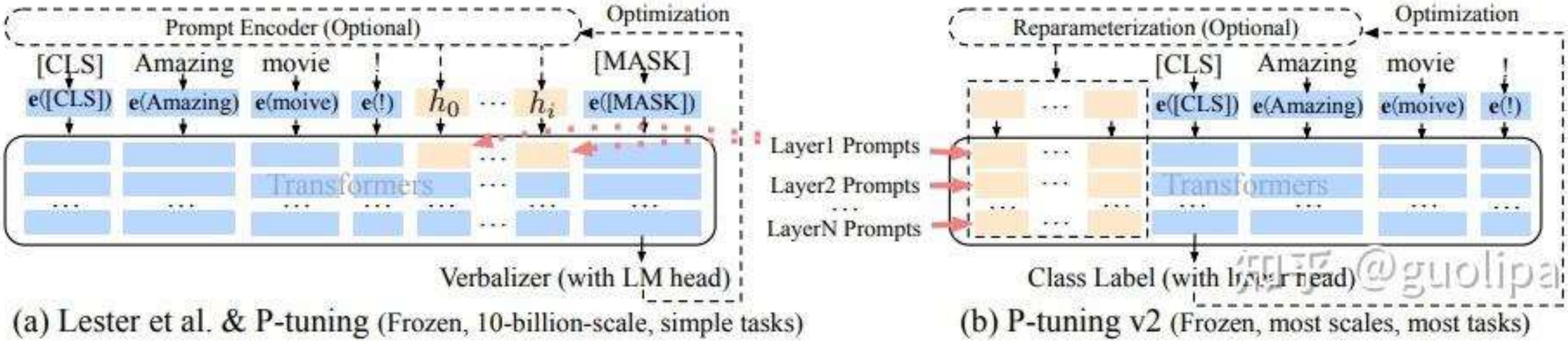
- **Prefix-Tuning 在每一层都添加可训练参数**
- **P-Tuning 只在 embedding 层增加参数**



知乎 @gzhlipei

扫码了解更多

大模型中的P-Tuning



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

(b) P-tuning v2 (Frozen, most scales, most tasks)

知乎 @guolipa

扫码了解更多

实验将五种方法进行对比，包括：Fine-Tuning (全量微调)、Bias-only or BitFit（只训练偏置向量）、Prefix-embedding tuning (PreEmbed，上文介绍的 Prefix Tuning 方法，只优化 embedding 层的激活)、Prefix-layer tuning (PreLayer，Prefix Tuning 方法，优化模型所有层的激活)、Adapter tuning（不同的 Adapter 方法：$\text{Adapter}^{\text{H}}$[10]、$\text{Adapter}^{\text{L}}$[11]、$\text{Adapter}^{\text{P}}$[12]、$\text{Adapter}^{\text{L}}$、$\text{Adapter}^{\text{D}}$[13]）

实验结果以 LoRA 在 GPT-3 175B 上的验证分析为例。如下表所示，**LoRA 在三个数据集上都能匹配或超过微调基准，证明了 LoRA 方法的有效性。**

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

扫码了解更多