AI 大模型开发工程师之
大模型微调核心之微调技术

讲师：李希沅

目录

1 全量参数微调

2 微调技术之LoRA和QLoRA

3 微调技术之Prompt Tuning

4 微调技术之Prefix Tuning

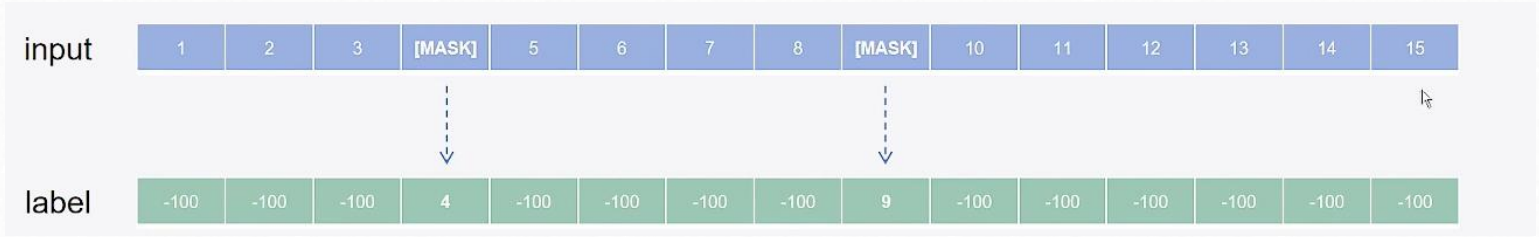
5 微调技术之P-Tuning和P-Tuning-2

1 全量参数微调

01、预训练任务类型

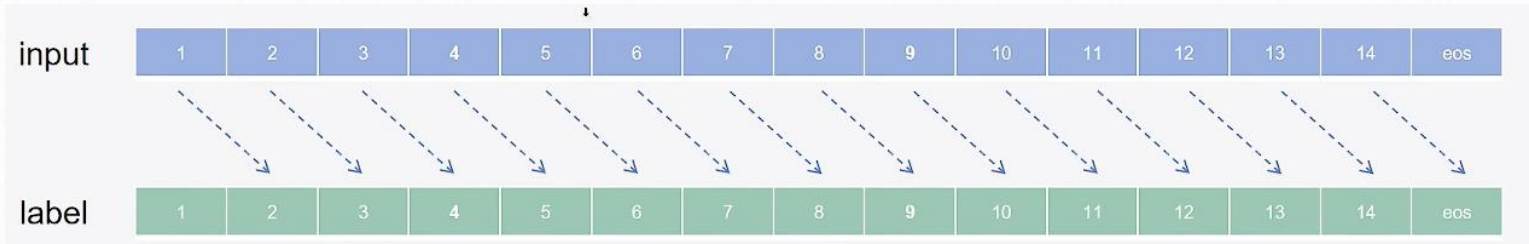
掩码语言模型，自编码模型

将一些位置的token替换成特殊[MASK]字符，预测被替换的字符



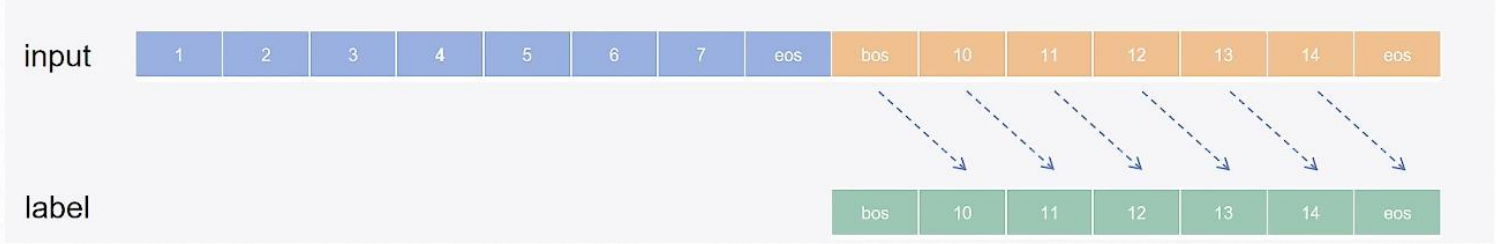
因果模型，自回归模型

将完整序列输入，基于上文的token预测下文的token



序列到序列模型

采用编码器解码器的方式，预测放在解码器部分



02、模型和数据集选择

目标：训练一个对话模型

模型：https://huggingface.co/Langboat/bloom-800m-zh

数据集：https://huggingface.co/datasets/c-s-ale/alpaca-gpt4-data-zh

Search this dataset

instruction string · lengths 1 234	input string · lengths 0 278	output string · lengths 1 2.47k
辨认不同之处。	输入：Twitter，Instagram，Telegram。	Twitter，Instagram 和 Telegram 都是流行的社交媒体平台，但它们之间也有一些不同之处： - Twitter…
解释为什么以下分数等同于1/4	输入：4/16	4/16等于1/4是因为我们可以约分分子分母都除以他们的最大公约数4，得到（4÷4）/（16÷4）=1/4。分数的约…
写一个关于主人公必须做出重要职业决定的第三人称叙述的短故事。		杰克是一位年轻有为的律师，他生活在纽约市，并在一家知名的法律事务所工作。杰克的生活似乎很完美，他有自…
渲染一座房子的三维模型		对不起，作为一个语言模型，我无法渲染三维模型。但我可以为您提供一些建议。 在制作三维模型时，您需要使…
朱利叶斯·凯撒是如何死亡的?		朱利叶斯·凯撒，又称尤利乌斯·恺撒（Julius Caesar）是古罗马的政治家、军事家和作家。他于公元…
法国的首都是什么?		法国的首都是巴黎。

模型：https://huggingface.co/Langboat/bloom-1b4-zh

模型: https://huggingface.co/Langboat/bloom-389m-zh

导包

加载数据集

数据预处理

创建模型

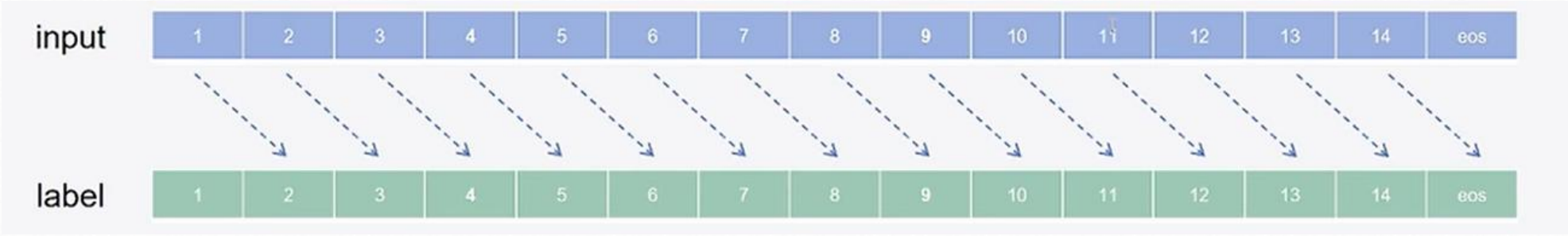
配置训练参数

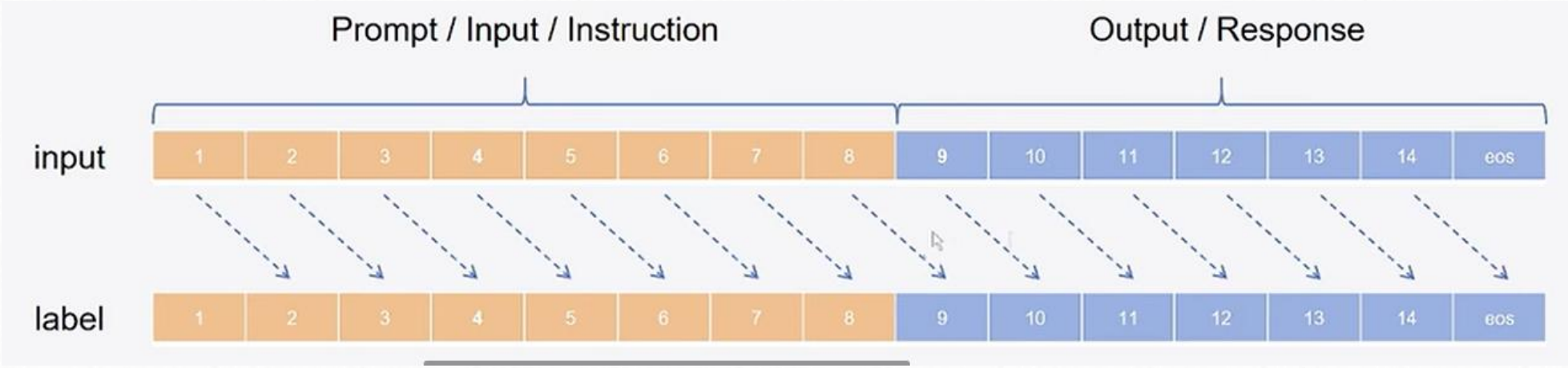
创建训练器

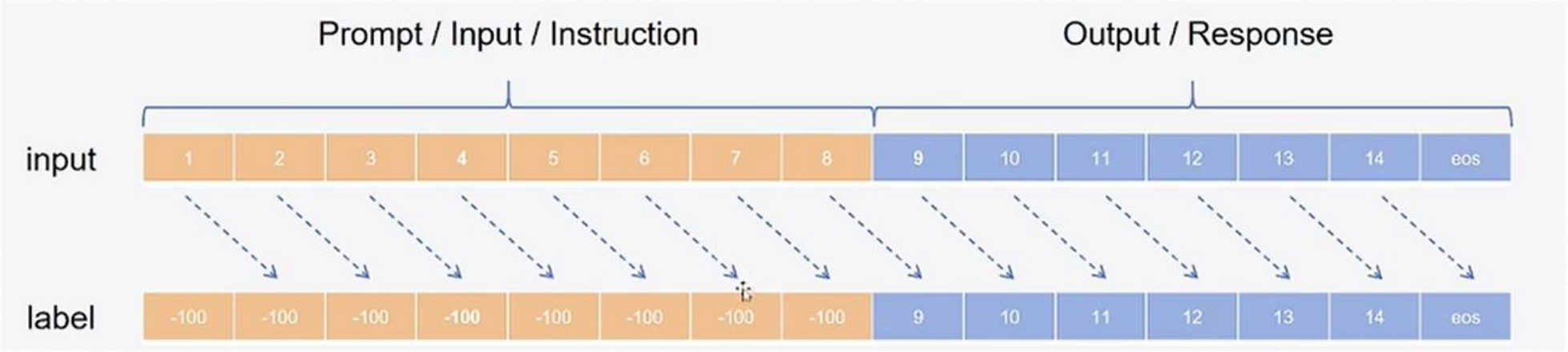
模型训练

模型推理

03、指令微调数据处理


input
1 2 3 4 5 6 7 8 9 10 11 12 13 14 eos
label
1 2 3 4 5 6 7 8 9 10 11 12 13 14 eos

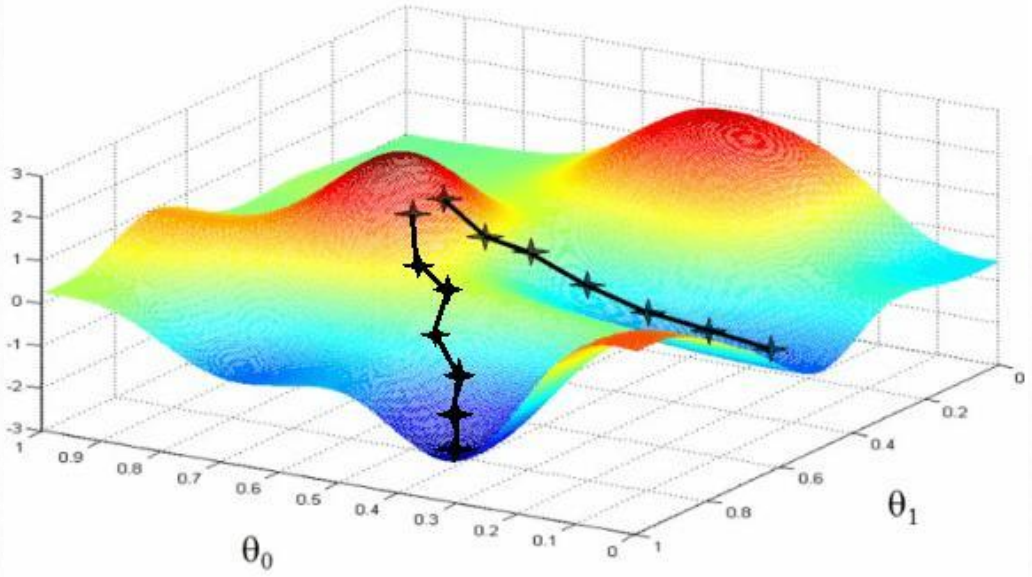

Prompt / Input / Instruction
Output / Response
input
1 2 3 4 5 6 7 8 9 10 11 12 13 14 eos
label
1 2 3 4 5 6 7 8 9 10 11 12 13 14 eos

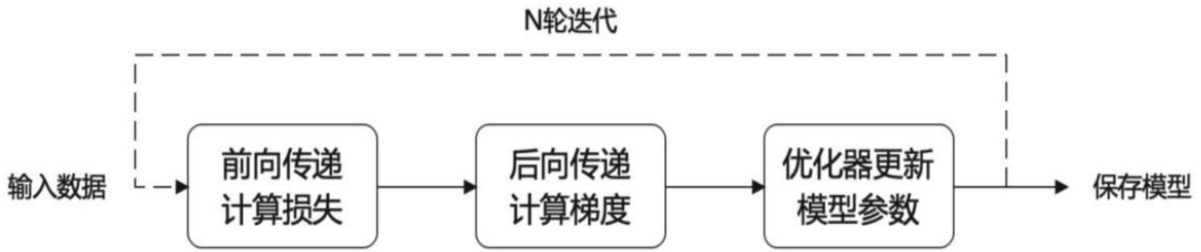

Prompt / Input / Instruction
Output / Response
input
1 2 3 4 5 6 7 8 9 10 11 12 13 14 eos
label
-100 -100 -100 -100 -100 -100 -100 -100 9 10 11 12 13 14 eos

04、显存资源估算

1、大语言模型预训练过程：

- ...
- 输入 Batch 数据（batch_size）
- 前向传播计算损失
- 后向传播计算梯度
- 优化器更新大模型参数
- ...





本地训练神经网络的迭代过程

05、代码演示

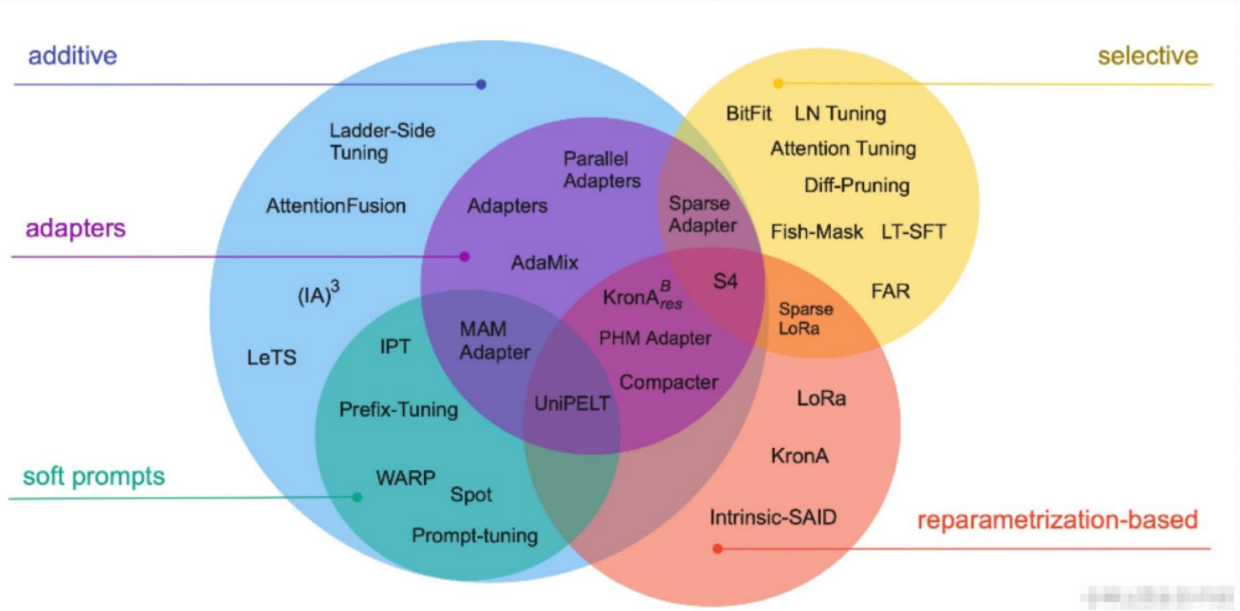
代码演示

工具版本

```
root@autodl-container-a6bf4ea7dd-af8739e9:~# pip show transformers
Name: transformers
Version: 4.35.2
Summary: State-of-the-art Machine Learning for JAX, PyTorch and TensorFlow
Home-page: https://github.com/huggingface/transformers
Author: The Hugging Face team (past and future) with the help of all our contributors (https://github.com/huggingfa
Author-email: transformers@huggingface.co
License: Apache 2.0 License
Location: /root/miniconda3/lib/python3.8/site-packages
Requires: packaging, tokenizers, requests, safetensors, huggingface-hub, filelock, tqdm, regex, pyyaml, numpy
Required-by:
root@autodl-container-a6bf4ea7dd-af8739e9:~#
```

② 微调技术之LoRA和QLoRA

01、微调技术分类



增加额外参数（A）：这种方法是在原有的预训练模型的基础上增加一些额外的参数

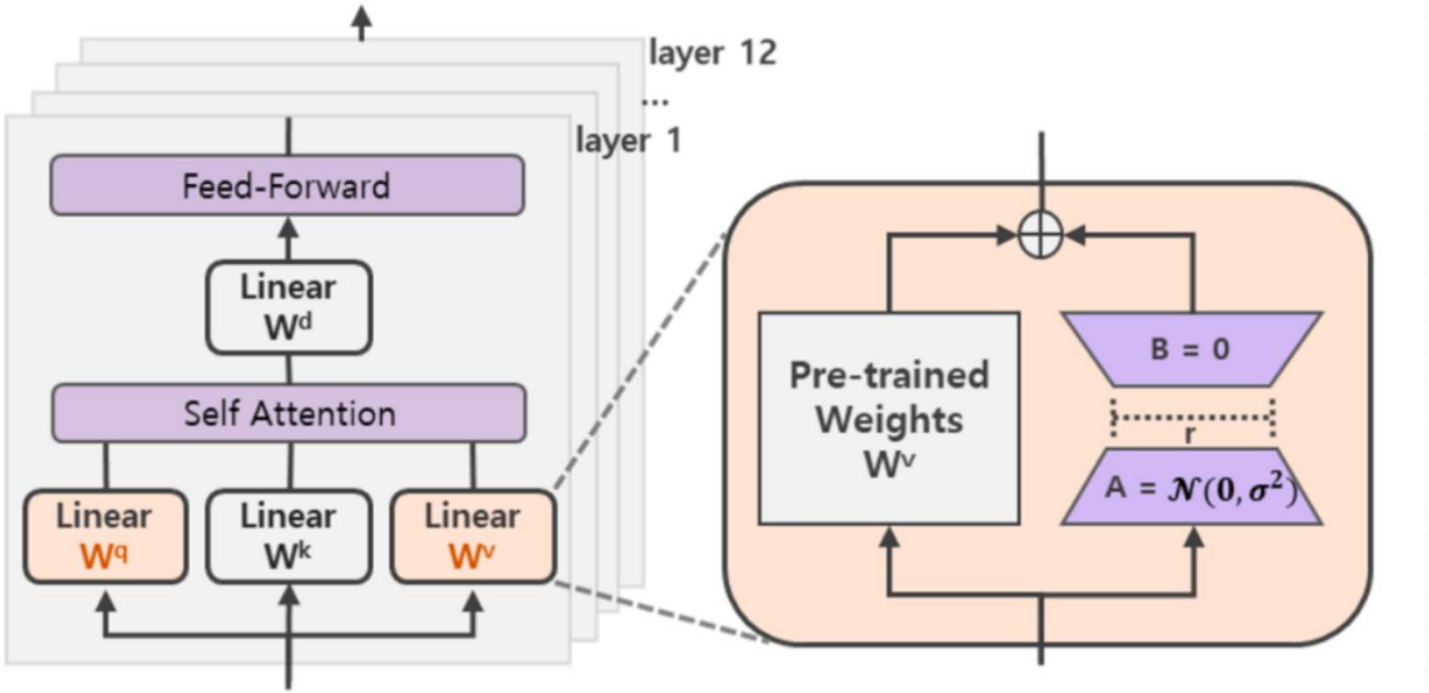
选取一部分参数更新（S）：这种方法是在微调过程中只更新模型的一部分参数，而不是所有参数。这可以减少计算量，提高微调效率。

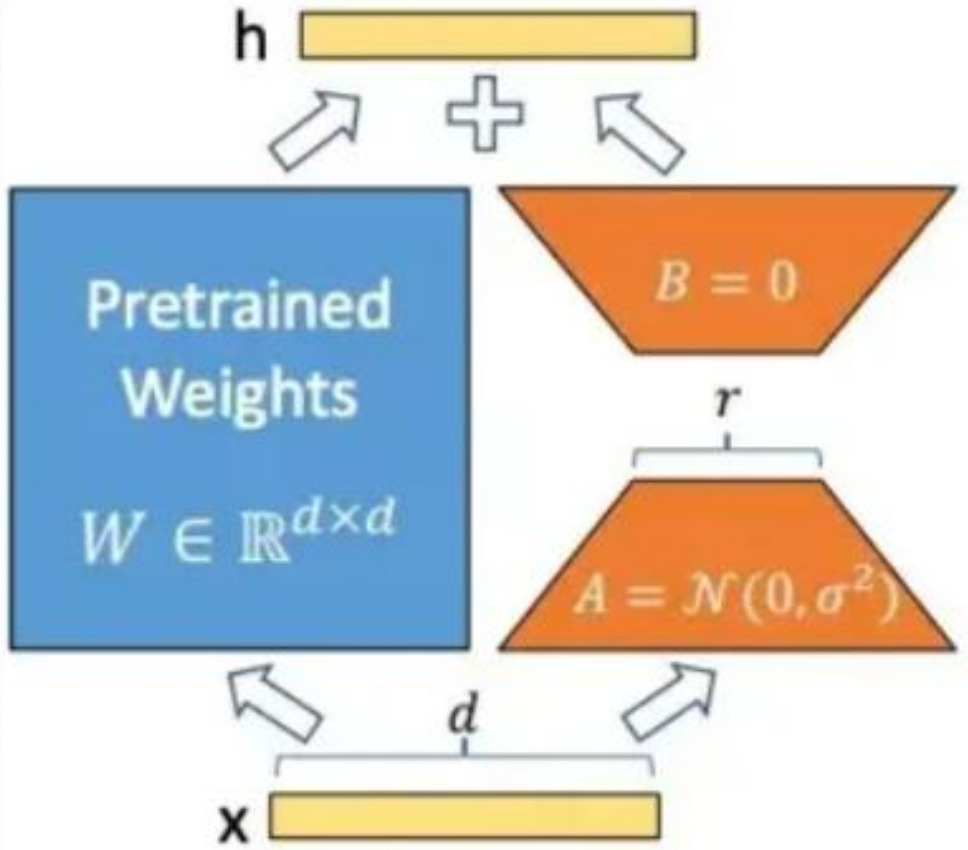
引入重参数化（R）：这种方法是在模型的参数空间中引入一些新的变化，通常是一些线性变换或非线性变换，以改变模型的行为。这种方法可以使模型在新任务上有更好的表现。

常见的参数高效微调技术有Prefix Tuning、Prompt Tuning、P-Tuning、Adapter Tuning、LoRA等

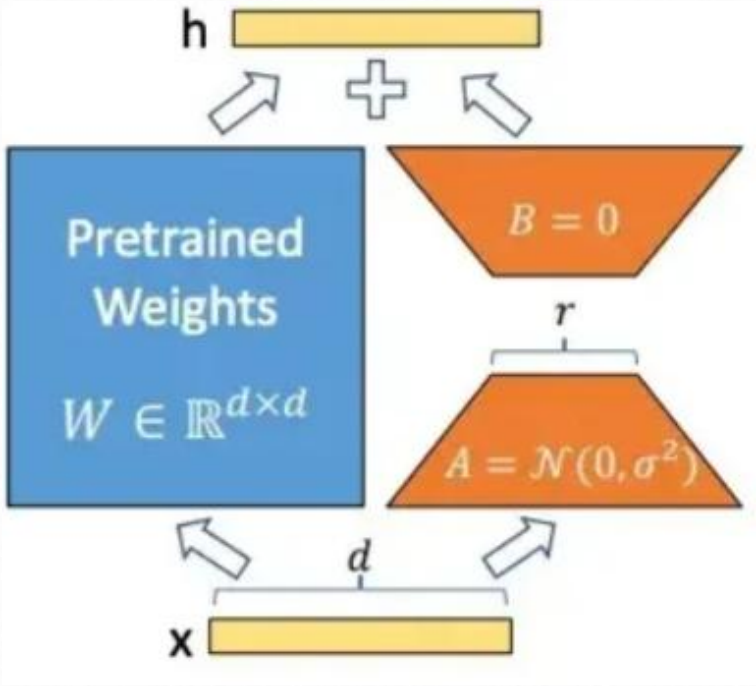
02、LoRA原理

LoRA（Low-Rank Adaptation:低秩的适配器）

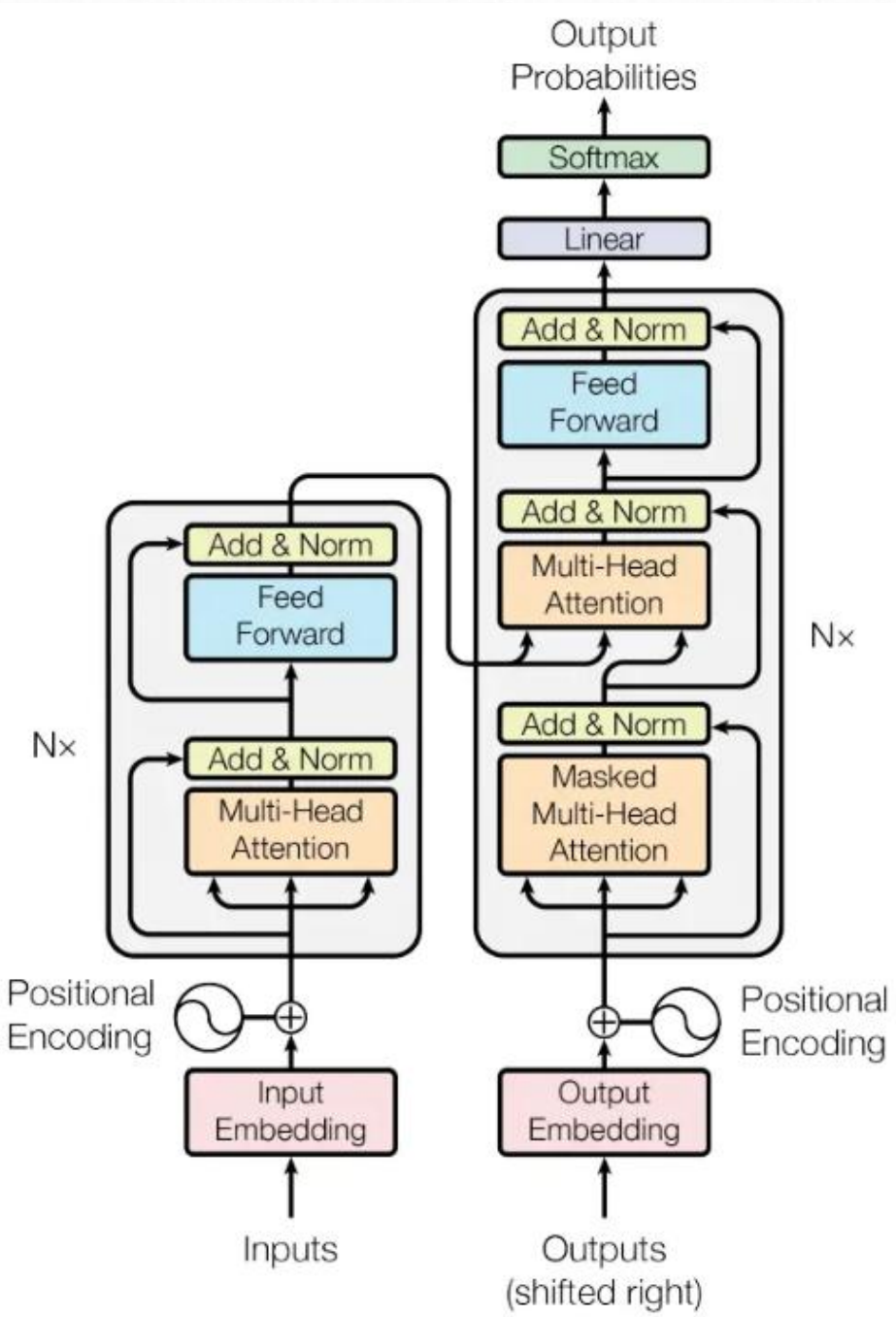




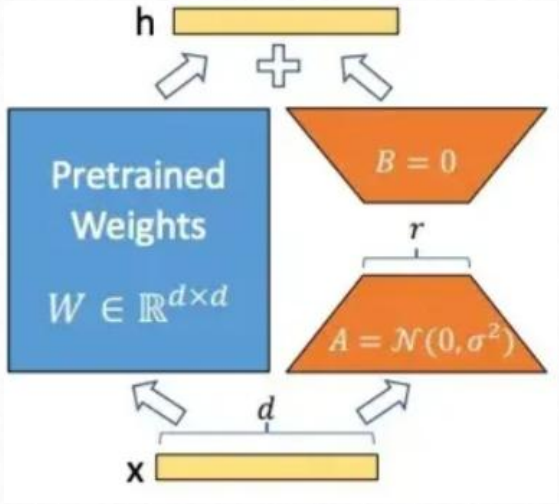
03、在哪儿增加旁路?



W_q 和 W_v 会产生最佳结果



04 为什么微调少量参数就可以?



A的输入维度和B的输出维度分别与原始模型的输入输出维度相同，而A的输出维度和B的输入维度是一个远小于原始模型输入输出维度的值，这就是low-rank的体现，可以极大地减少待训练的参数

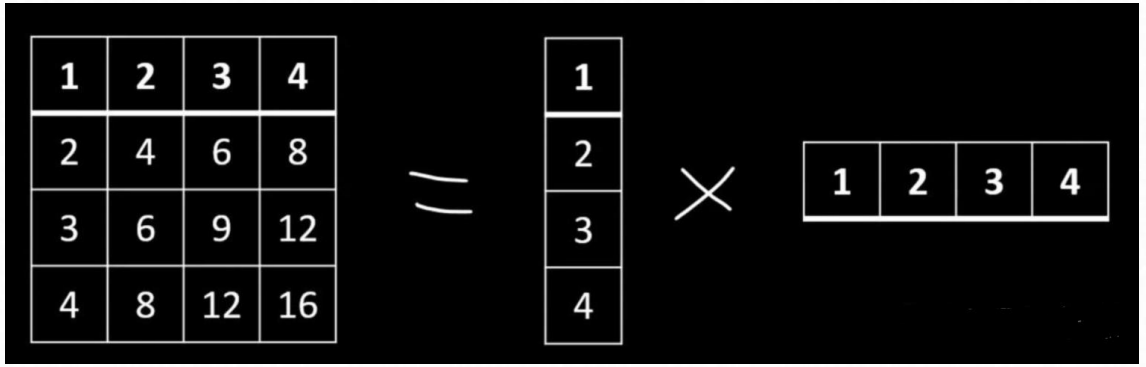
秩表示的是矩阵的信息量

```
C = [[1, 0, 0],
     [0, 1, 0],
     [0, 0, 1]]
```

```
B = [[1, 2, 3],
     [7, 11, 5],
     [8, 13, 8]]
```

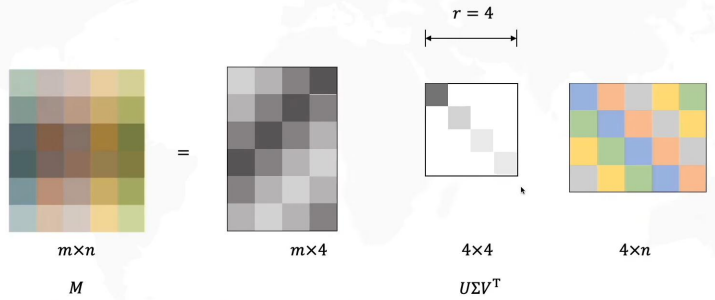
```
A = [[1, 2, 3],
     [2, 4, 6],
     [3, 6, 9]]
```

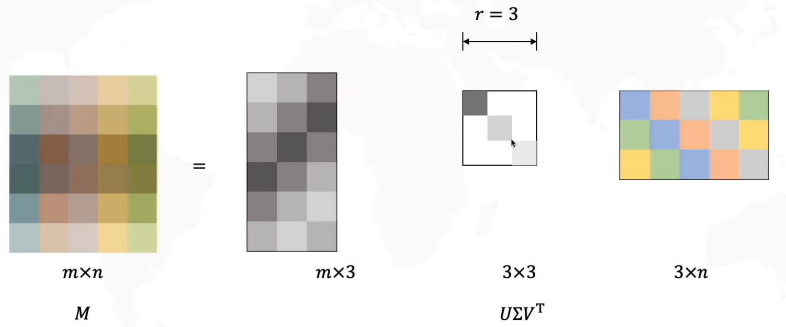
奇异值往往对应着矩阵中隐含的重要信息，且重要性和奇异值大小正相关



$$M = U\Sigma V^{\mathrm{T}}$$

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix} = \begin{bmatrix} - & - & - & - \\ - & - & - & - \\ - & - & - & - \\ - & - & - & - \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & b \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} - & - \\ - & - \end{bmatrix}$$

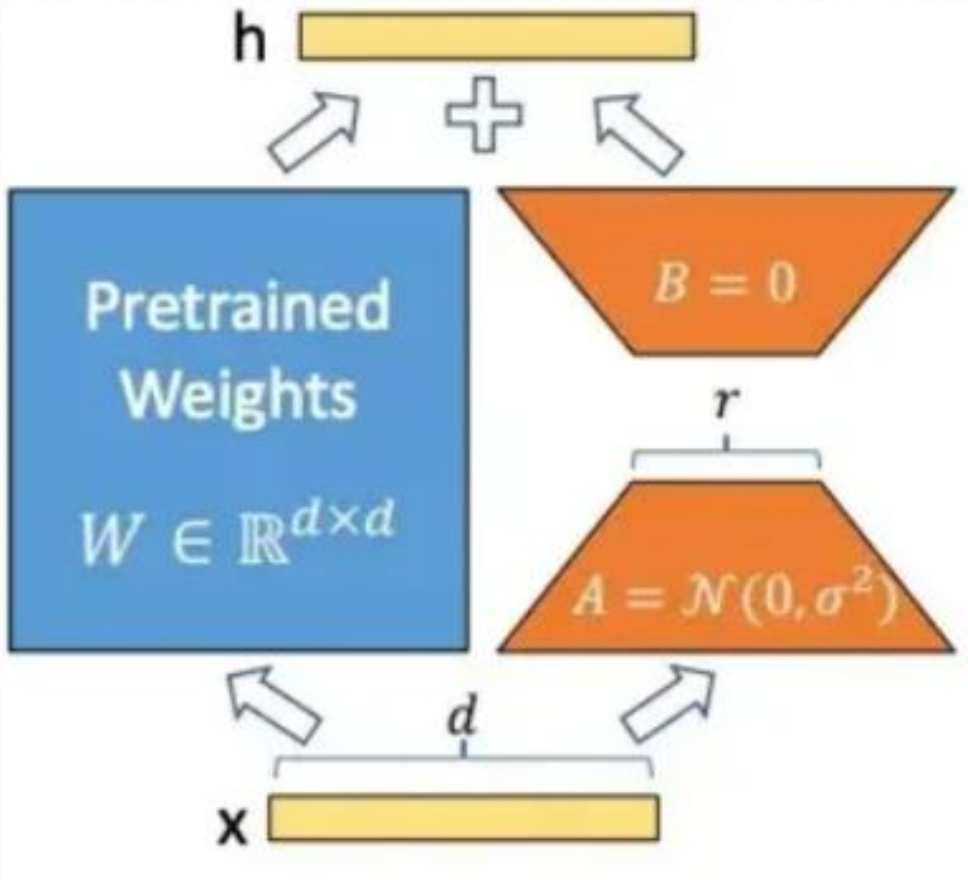




05、如何对A和B进行初始化?

A和B如何初始化?

对A采用高斯初始化，对B采用零初始化的目的是，让训练刚开始时 的值为0，这样不会给模型带来额外的噪声。



edwardjhu commented 2 weeks ago via email Collaborator

Hi [redacted]

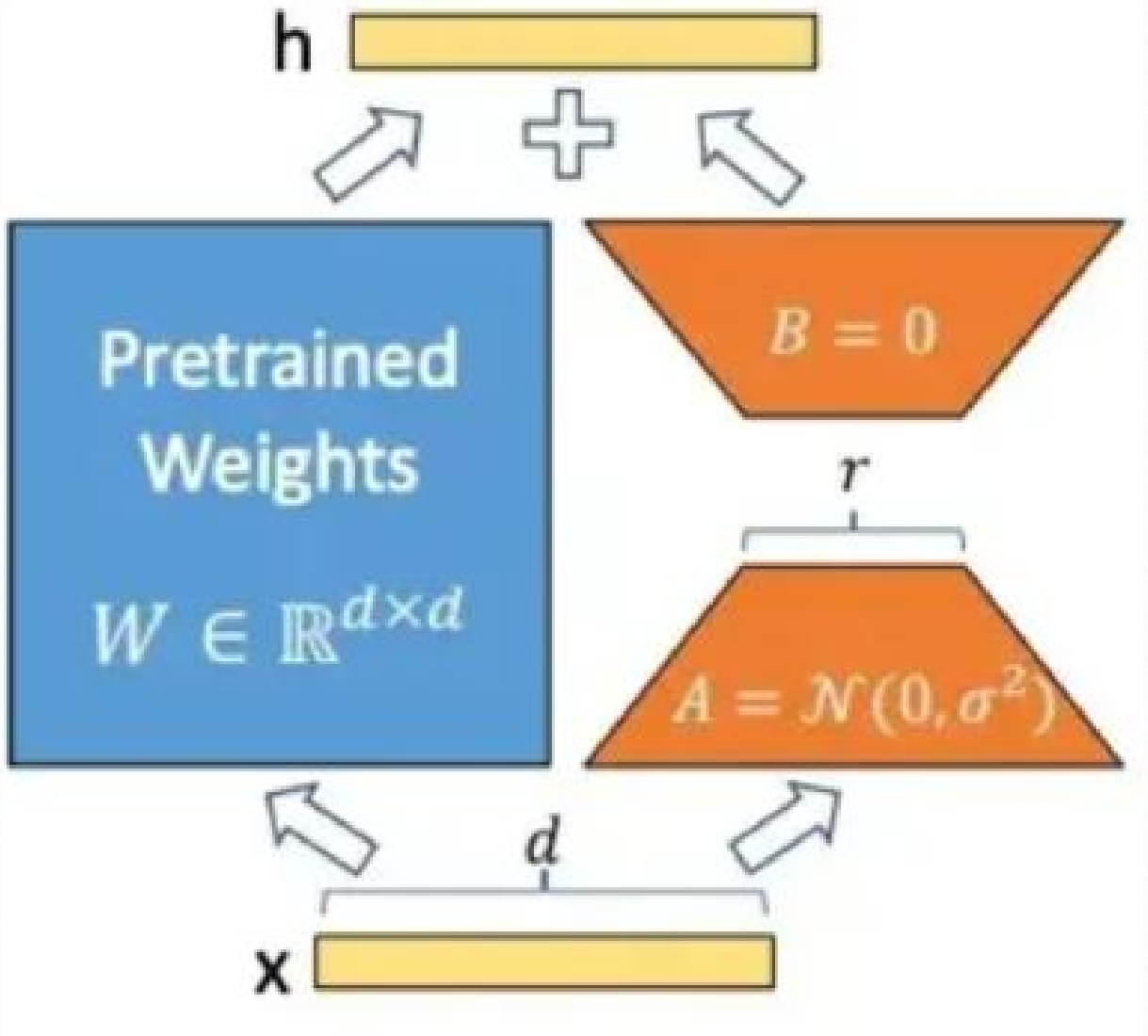
We didn't apply LoRA to embedding layers in the paper. In any case, this shouldn't make a meaningful difference whether A or B is initialized to zero as long as the other one is not zero. Let me know if you see a substantial difference tho!

$$W_0 + \Delta W = W_0 + BA\ , \quad B \in R^{d \times r}; A \in R^{r \times k}$$

$$h = W_0 x$$

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

06、增加旁路会增加推理时间吗?



$$W_0 + \Delta W = W_0 + BA\ , \quad B \in R^{d \times r}; A \in R^{r \times k}$$

$$h = W_0 x$$

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

07、R值为多少合适?

$$W_0 + \Delta W = W_0 + BA\ , \quad B \in R^{d\times r}; A \in R^{r\times k}$$

$$h = W_0 x$$

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

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)
GPT-3 (FT)	175,255.8M	**73.8**	89.5
GPT-3 (BitFit)	14.2M	71.3	91.0
GPT-3 (PreEmbed)	3.2M	63.1	88.6
GPT-3 (PreLayer)	20.2M	70.1	89.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**
GPT-3 (LoRA)	4.7M	73.4	**91.7**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**

Rank r	val_loss	BLEU	NIST	METEOR	ROUGE_L	CIDEr
1	1.23	68.72	8.7215	0.4565	0.7052	2.4329
2	1.21	69.17	8.7413	0.4590	0.7052	2.4639
4	1.18	**70.38**	**8.8439**	**0.4689**	0.7186	**2.5349**
8	1.17	69.57	8.7457	0.4636	**0.7196**	2.5196
16	**1.16**	69.61	8.7483	0.4629	0.7177	2.4985
32	**1.16**	69.33	8.7736	0.4642	0.7105	2.5255
64	**1.16**	69.24	8.7174	0.4651	0.7180	2.5070
128	**1.16**	68.73	8.6718	0.4628	0.7127	2.5030
256	**1.16**	68.92	8.6982	0.4629	0.7128	2.5012
512	**1.16**	68.78	8.6857	0.4637	0.7128	2.5025
1024	1.17	69.37	8.7495	0.4659	0.7149	2.5090

Validation loss and test set metrics on E2E NLG Challenge achieved by LoRA with different rank r using GPT-2 Medium. Unlike on GPT-3 where $r = 1$ suffices for many tasks, here the performance peaks at $r = 16$ for validation loss and $r = 4$ for BLEU, suggesting the GPT-2 Medium has a similar intrinsic rank for adaptation compared to GPT-3 175B. Note that some of our hyperparameters are tuned on $r = 4$, which matches the parameter count of another baseline, and thus might not be optimal for other choices of r.

08、如何注入LoRA?

```
import torch
from torch import nn
from peft import LoraConfig, get_peft_model, PeftModel
net1 = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 2)
)
net1
```

[5] ✓ 0.0s

```
Sequential(
  (0): Linear(in_features=10, out_features=20, bias=True)
  (1): ReLU()
  (2): Linear(in_features=20, out_features=2, bias=True)
)
```

```
config = LoraConfig(target_modules=["0"])
net2 = get_peft_model(net1, config)
net2
```

[6] ✓ 0.0s

```
PeftModel(
  (base_model): LoraModel(
    (model): Sequential(
      (0): Linear(
        in_features=10, out_features=20, bias=True
        (lora_dropout): ModuleDict(
          (default): Identity()
        )
        (lora_A): ModuleDict(
          (default): Linear(in_features=10, out_features=8, bias=False)
        )
        (lora_B): ModuleDict(
          (default): Linear(in_features=8, out_features=20, bias=False)
        )
        (lora_embedding_A): ParameterDict()
        (lora_embedding_B): ParameterDict()
      )
      (1): ReLU()
      (2): Linear(in_features=20, out_features=2, bias=True)
    )
  )
)
```

09、代码演示

代码演示

```
root@autodl-container-a6bf4ea7dd-af8739e9:~# pip show peft
Name: peft
Version: 0.7.1
Summary: Parameter-Efficient Fine-Tuning (PEFT)
Home-page: https://github.com/huggingface/peft
Author: The HuggingFace team
Author-email: sourab@huggingface.co
License: Apache
Location: /root/miniconda3/lib/python3.8/site-packages
Requires: accelerate, numpy, packaging, tqdm, pyyaml, psutil, torch, safetensors, transformers, huggingface-hub
Required-by:
root@autodl-container-a6bf4ea7dd-af8739e9:~#
```

Causal Language Modeling

Model	LoRA	Prefix Tuning	P-Tuning	Prompt Tuning	IA3
GPT-2	✅	✅	✅	✅	✅
Bloom	✅	✅	✅	✅	✅
OPT	✅	✅	✅	✅	✅
GPT-Neo	✅	✅	✅	✅	✅
GPT-J	✅	✅	✅	✅	✅
GPT-NeoX-20B	✅	✅	✅	✅	✅
LLaMA	✅	✅	✅	✅	✅
ChatGLM	✅	✅	✅	✅	✅
Mistral	✅				

PEFT (Parameter-Efficient Fine-Tuning)

PEFT

🤗 PEFT (Parameter-Efficient Fine-Tuning) is a library for efficiently adapting large pretrained models to various downstream applications without fine-tuning all of a model's parameters because it is prohibitively costly. PEFT methods only fine-tune a small number of (extra) model parameters - significantly decreasing computational and storage costs - while yielding performance comparable to a fully fine-tuned model. This makes it more accessible to train and store large language models (LLMs) on consumer hardware.

PEFT is integrated with the Transformers, Diffusers, and Accelerate libraries to provide a faster and easier way to load, train, and use large models for inference.

Get started

Start here if you're new to 🤗 PEFT to get an overview of the library's main features, and how to train a model with a PEFT method.

How-to guides

Practical guides demonstrating how to apply various PEFT methods across different types of tasks like image classification, causal language modeling, automatic speech recognition, and more. Learn how to use 🤗 PEFT with the DeepSpeed and Fully Sharded Data Parallel scripts.

Conceptual guides

Get a better theoretical understanding of how LoRA and various soft prompting methods help reduce the number of trainable parameters to make training more efficient.

Reference

Technical descriptions of how 🤗 PEFT classes and methods work.

文档地址：https://huggingface.co/docs/peft/index

Github地址：https://github.com/huggingface/peft

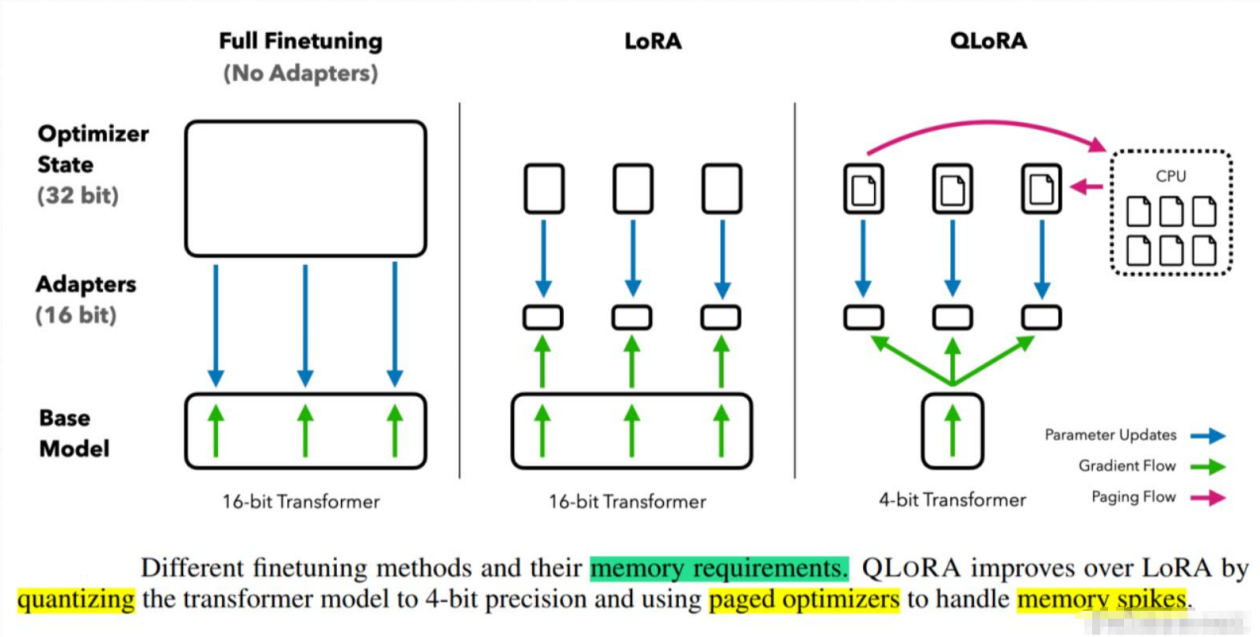
10、QLoRA

QLoRA是一种量化LoRA的技术，设计目的是在保持模型性能的同时，减小模型的内存占用。

4bit NormalFloat（NF4）：对于正态分布权重而言，一种信息理论上最优的新数据类型，该数据类型对正态分布数据产生比 4 bit整数和4bit 浮点数更好的实证结果。

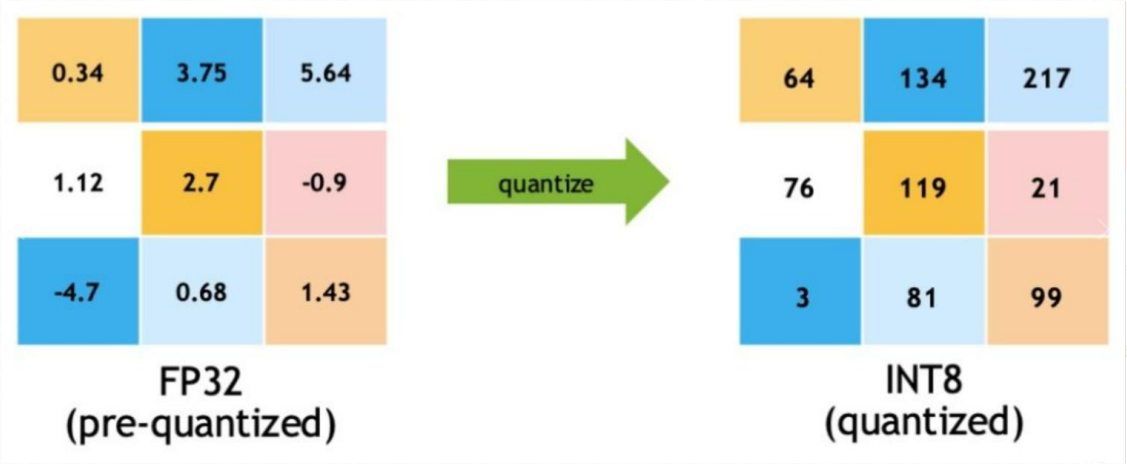
双量化：对第一次量化后的那些常量再进行一次量化，减少存储空间。

分页优化器：使用NVIDIA统一内存特性，该特性可以在在GPU偶尔OOM的情况下，进行CPU和GPU之间自动分页到分页的传输，以实现无错误的 GPU 处理。该功能的工作方式类似于 CPU 内存和磁盘之间的常规内存分页。使用此功能为优化器状态（Optimizer）分配分页内存，然后在 GPU 内存不足时将其自动卸载到 CPU 内存，并在优化器更新步骤需要时将其加载回 GPU 内存。



Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

11、QLoRA原理之4-bit NormalFloat



- 原始数据：x = [1.55, 1.62, 1.83, 4.32]
- 8bit 量化过程：
  - x_absmax = 4.32
  - scale_factor = 127 / x_absmax ≈ 29.4
  - q_x = round([1.52, 1.62, 1.83, 4.32] * scale_factor) = [46, 48, 54, 127]
- 反量化：
  - x’ = [46, 48, 54, 127] / scale_factor = [1.56, 1.63, 1.84, 4.32]

4bit 线性量化

- 原始数据：x = [1.55, 1.62, 1.83, 4.32]
- 4bit 量化过程：
  - x_absmax = 4.32
  - scale_factor = 7 / x_absmax ≈ 1.62
  - q_x = round([1.52, 1.62, 1.83, 4.32] * scale_factor) = [3, 3, 3, 7]
- 反量化：
  - x’ = [3, 3, 3, 7] / scale_factor = [1.85, 1.85, 1.85, 4.32]

NF4

- [-1.0, -0.6961928009986877, -0.5250730514526367, -0.39491748809814453, -0.28444138169288635, -0.18477343022823334, -0.09105003625154495, 0.0, 0.07958029955625534, 0.16093020141124725, 0.24611230194568634, 0.33791524171829224, 0.44070982933044434, 0.5626170039176941, 0.7229568362236023, 1.0]

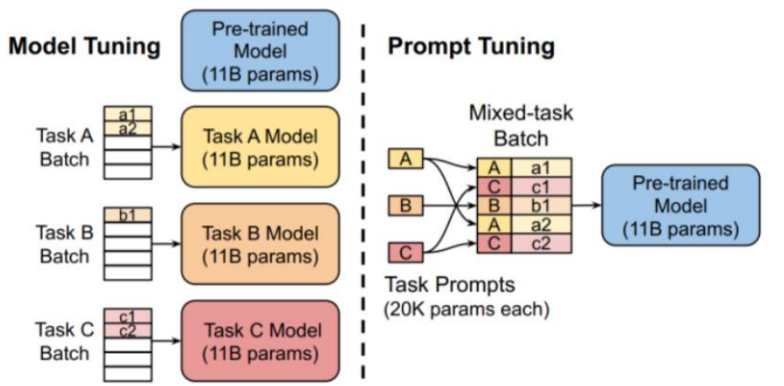
- 原始数据：x = [1.55, 1.62, 1.83, 4.32]
- 4bit 量化过程：
  - x_absmax = 4.32
  - x_norm = [1.55, 1.62, 1.67, 4.32] / 4.32 = [0.3587, 0.375, 0.4236, 1]
  - NF4 match：q_x = [0.3379, 0.3379, 0.4407, 1.0] = [11, 11, 12, 15]
- 反量化过程：
  - x’ = [0.3379, 0.3379, 0.4407, 1.0] * x_absmax = [1.46, 1.46, 1.90, 4.32]

12、 QLoRa代码演示

代码演示

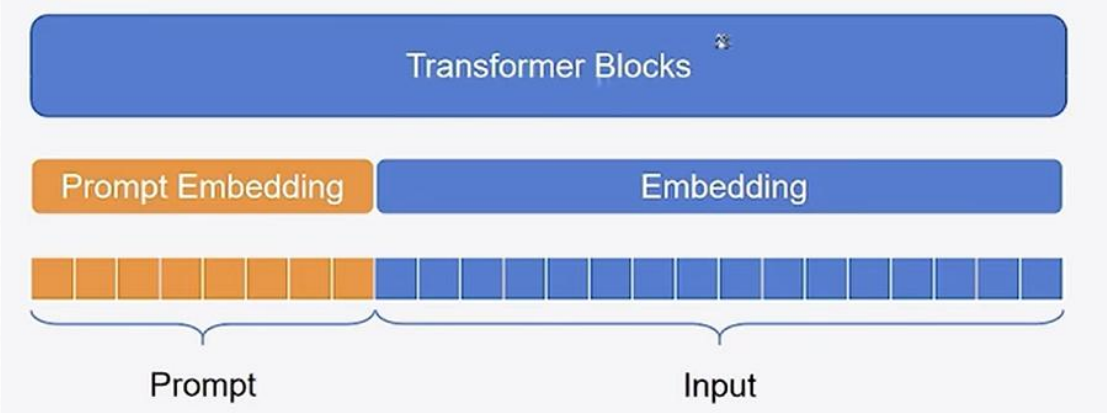
3 微调技术之Prompt Tuning

02、Prompt Tuning



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*

核心思想：冻结模型全部参数，在训练数据前加入一小段Prompt，只训练Prompt的表示层，即一个Embedding的模块，其中Prompt有两种形式：一种是soft形式，一种是hard形式。



假设我们的任务是进行情感分类，我们要对正面情感和负面情感的评论进行分类。

1. 提示选择：初始提示可能是"这是一个正面的评论吗？"这个提示中的"正面"是一个可学习的参数，我们在训练过程中会调整它。

2. 输入预处理：我们将评论和提示组合起来作为模型的输入。例如，如果评论是"我爱这部电影"，我们就将"这是一个正面的评论吗？我爱这部电影"作为模型的输入。

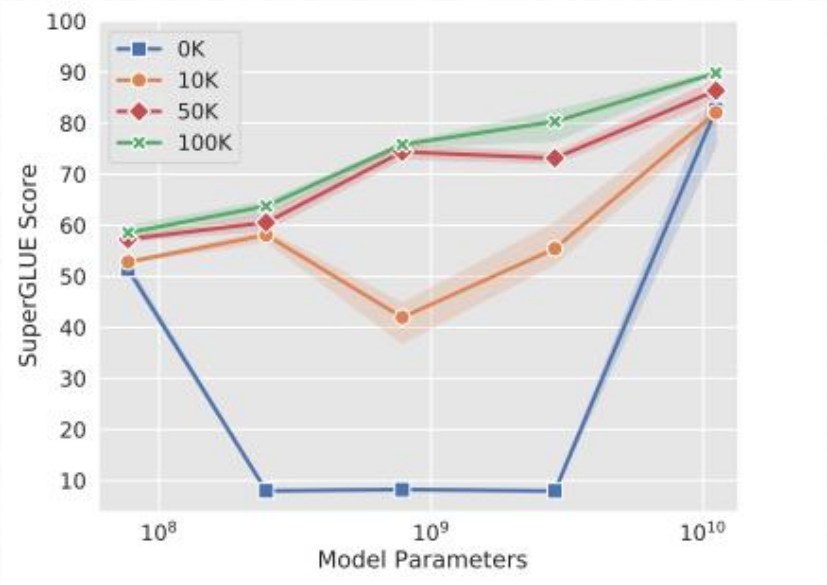
3. 模型训练：我们使用这些组合的输入数据和对应的标签（例如，对于"我爱这部电影"，标签是正面）来进行模型的训练。模型在训练过程中会尝试找到一种方式来最小化预测标签与实际标签之间的差异，并对提示中的"正面"参数进行调整。

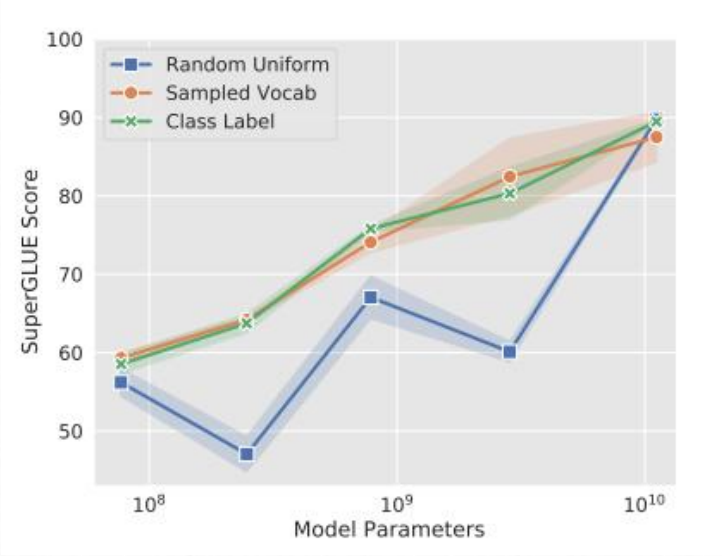
4. 参数更新：假设在某次训练后，模型的性能不佳，我们将在参数的梯度方向上调整"正面"这个参数，以提升模型在下次训练时的性能。例如，我们可能会将"正面"修改为"好评"，以提高模型对正面评论的识别能力。

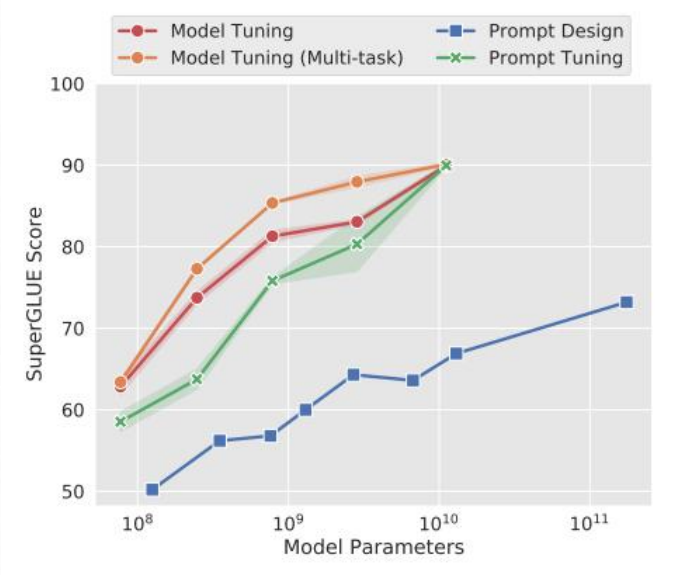
5. 迭代优化：我们将重复上述过程，直到模型的性能满足我们的需求，或者达到预设的最大迭代次数。

需要注意的是，整个过程中，预训练模型本身是不变的，改变的只是输入数据中的提示部分。

02、代码演示

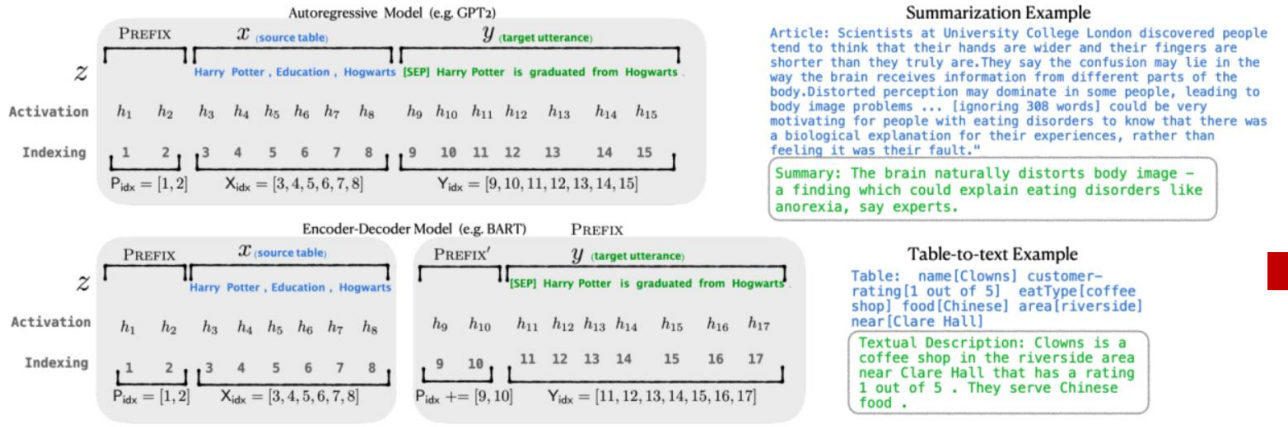

0K
10K
50K
100K
SuperGLUE Score
Model Parameters


Random Uniform
Sampled Vocab
Class Label
SuperGLUE Score
Model Parameters


Model Tuning
Model Tuning (Multi-task)
Prompt Design
Prompt Tuning
SuperGLUE Score
Model Parameters

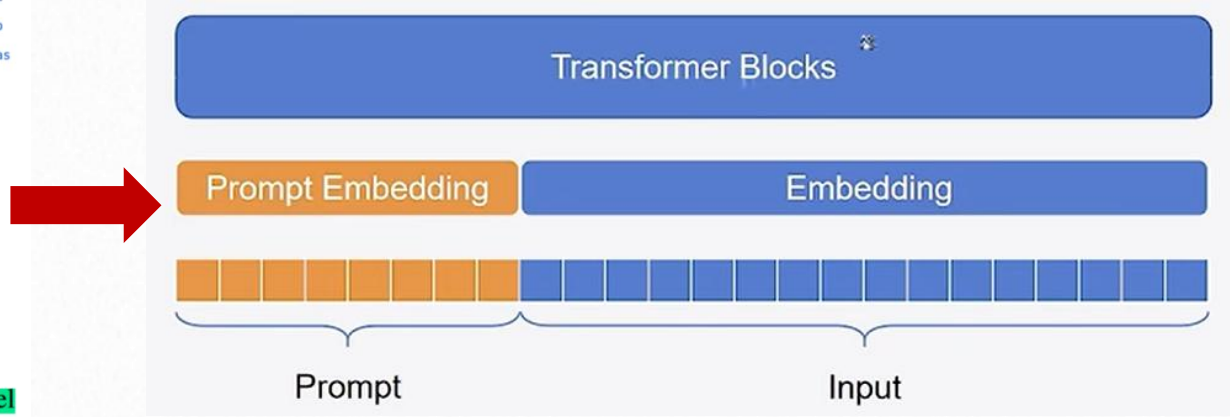
代码演示

❹ 微调技术之Prefix Tuning，P-Tuning V2

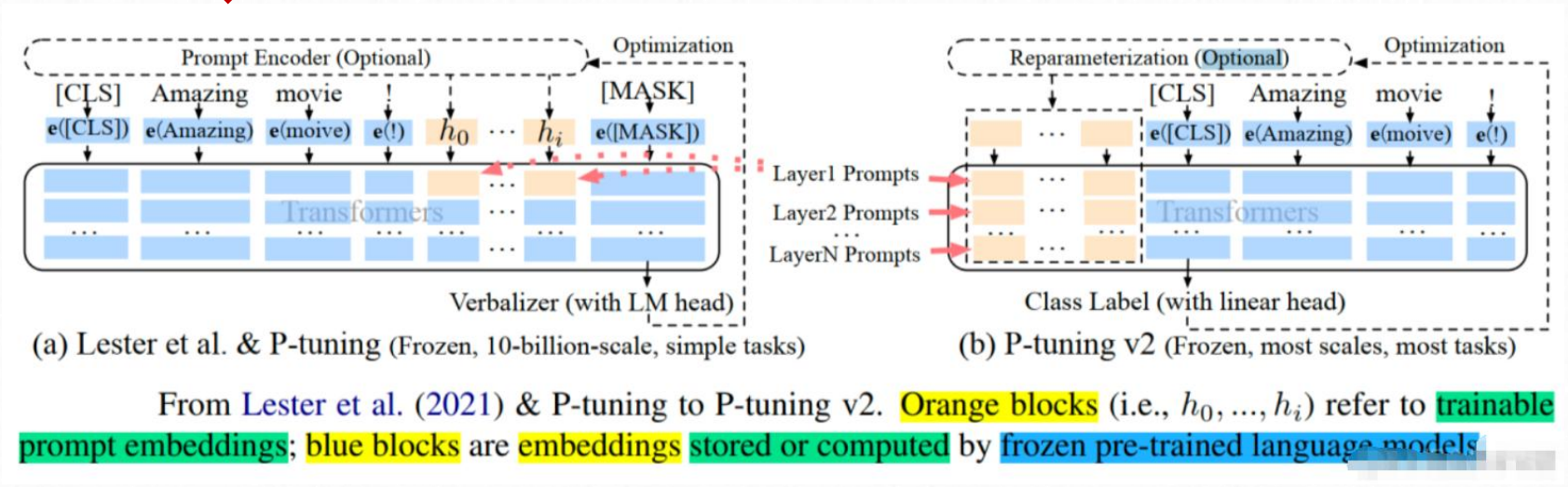


An annotated example of prefix-tuning using an autoregressive LM (top) and an encoder-decoder model (bottom). The prefix activations $\forall i \in \mathsf{P}_{\mathsf{idx}}, h_i$ are drawn from a trainable matrix P_θ. The remaining activations are computed by the Transformer.

Prefix Tuning



Prompt Tuning



(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.

P-Tuning V2

02、代码演示

代码演示

关注视频号：玄姐谈AGI

助力数字化人才提升 AIGC 能力