Fine-tuning微调艺术

>> 今天的学习目标

Fine-tuning微调艺术

- 高效微调方法概述

模型微调的类型

高效微调的方法

- LoRA的数学原理

LoRA算法的核心假设

矩阵分解与猜你喜欢

SVD矩阵分解

- 微调数据准备

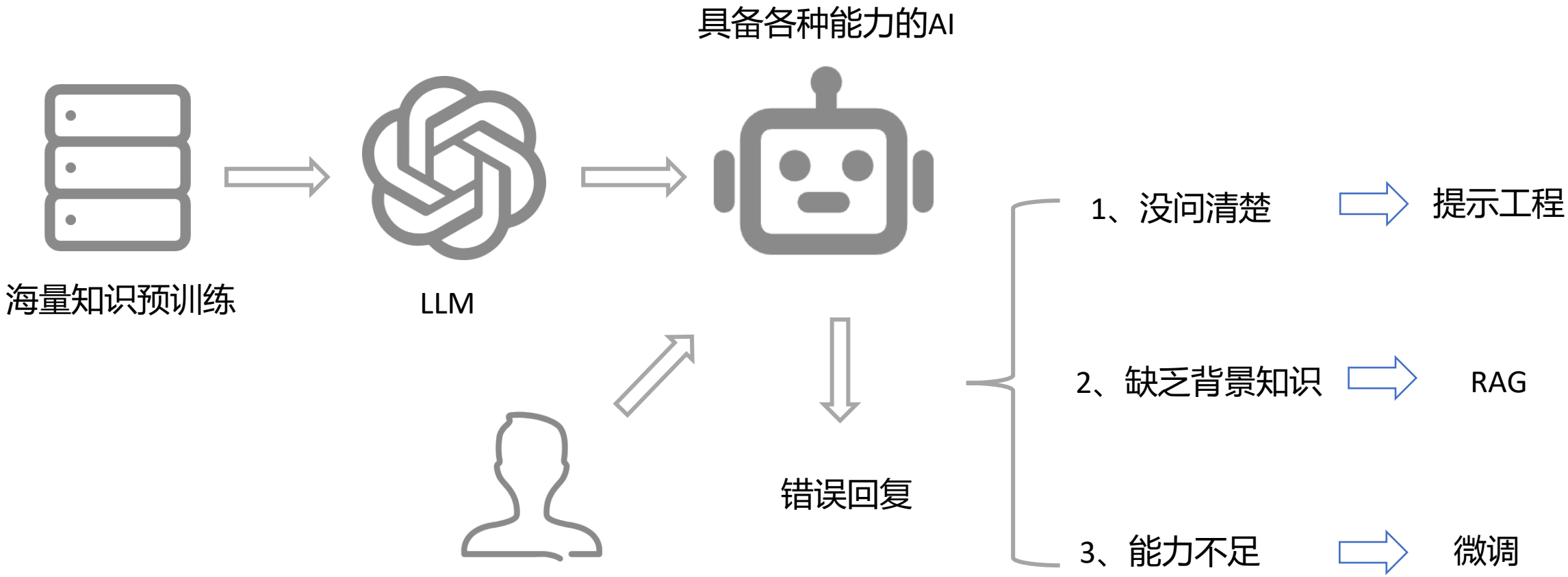
数据质量与数量要求

不同模型尺寸与场景的数据需求

- 硬件需求与显存计算

大模型应用开发

Thinking：提示工程 VS RAG VS 微调，什么时候使用？



高效微调方法

模型微调

Thinking：做AI应用需要的微调和基座模型公司做的微调，一般都是什么区别？

基座模型公司（OpenAI、Google、 Meta、阿里、DeepSeek、月之暗面）

目标：通用语言能力 → 规模大（100B+）、数据通用（万亿 token）、成本高（千卡月级）。

技术：继续预训练（continue pre-training）+ 指令微调（instruction tuning）+ RLHF/PPO/DPO。

使用方式：公开权重或 API。

模型微调

应用开发者（企业 IT部门、创业公司、个人开发者）

目标：垂直领域、私域知识（医疗、法律、客服…）。

规模：6B–70B，LoRA/QLoRA 为主，数小时～数天即可。

数据：1k–1M 条高质量指令-回答对，业务本身产生。

交付形式：增量 LoRA 权重（几十 MB），可热插拔。

高效微调的方法

方法	可训练参数比例	核心思想	常用场景
Prompt Tuning	极低 (<0.01%)	【软提示】 冻结整个预训练模型，只在输入层添加一串可训练的“软提示”向量（即不是人能读的自然语言词），让模型自适应地理解任务。	模型规模非常大（>10B）时效果才好，资源极度受限的场景。
P-Tuning v1/v2	极低 (<0.1%~1%)	【可深度的软提示】 是Prompt Tuning的升级版。将可训练的“软提示”向量插入到每一层的输入中，而不仅仅是输入层，使提示更深入、效果更强。	自然语言理解（NLU）任务，如分类、阅读理解。在中小模型（~10B）上效果也比Prompt Tuning好。
Prefix Tuning	低 (~0.1%~3%)	【隐式前缀】 与P-Tuning类似，也是在每一层头部添加可训练向量。但它将这些向量视为虚拟的“前缀token”，通过一个更复杂的网络（MLP）生成，之后被冻结。	自然语言生成（NLG）任务，如对话、摘要、翻译。
LoRA	低-中 (~1%~10%)	【低秩更新】 冻结预训练模型权重。假设模型微调时的权重变化是低秩的，用两个小矩阵（降维再升维）的乘积来近似这个更新量。	几乎全能，是目前最流行、最通用的方法。尤其适合微调LLM和扩散模型（如Stable Diffusion）。
QLoRA	低-中 (~1%~10%)	【量化LoRA】 LoRA的内存高效版。先将预训练模型量化为4-bit以极大减少内存占用，然后再使用LoRA进行微调。保证了性能几乎无损。	资源受限的场景（如单张消费级显卡），想要微调极其巨大的模型（如65B）。

LoRA的数学原理

Lora原理

Thinking：LoRA 微调的核心思想是什么？

假设模型微调过程中的权重更新（ΔW）具有“内在的低秩特性”

1. 核心问题：微调一个巨大模型很费劲

像GPT、 Stable Diffusion这样的大模型有数十亿参数。

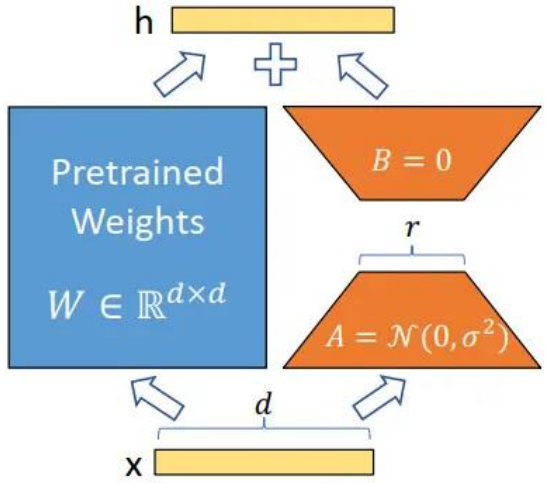
直接微调所有参数（全量微调）需要巨大的计算资源和存储空间，成本非常高。

2. LoRA 的聪明办法：不直接调原始权重

LoRA（Low-Rank Adaptation，低秩自适应）提出了一个巧妙的思路：

我们不改变模型原有的巨大权重矩阵 W。

我们在其旁边增加一个小的“旁路”，通过训练这个旁路来间接影响原始权重。



Lora原理

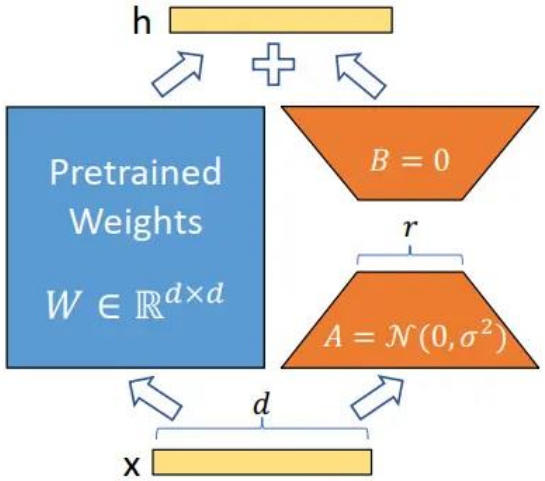
前向传播公式变为：h = Wx + BAx

W 是原始预训练权重，冻结不动。

A 和 B 是我们新引入的两个小的、低秩的矩阵。A负责降维，B负责升维。

BA 合在一起就构成了对原始权重的更新 ΔW。这个 ΔW 就是一个低秩矩阵。

微调时，我们只训练 A 和 B 这两个小矩阵，最后只需保存它们（文件很小），推理时再合并到 W 中即可。



Lora原理

Thinking：什么是“低秩特性”？

想象一个巨大的Excel表格（比如1000行 x 1000列），里面填满了数字。这个表格就是一个“矩阵”。

高秩矩阵：这个表格里的数据五花八门，每一行、每一列的信息都独一无二，没有明显的规律。你想简化它？没门！你必须原封不动地保存这100万个数字才能完整描述它。

低秩矩阵：表格里的数据高度相关，存在明显的“套路”。

例如：可能所有行都是第一行的倍数。

=> 你只需要保存很少的几行（或几列）基础数据，就能通过组合完美地重建出整个巨大的表格。

思考：比如一份全球气温报告。你不需要保存每个城市的每分钟数据。你只需要知道“纬度”、“季节”等几个核心因素（低维基础），就能很好地推测出任意地方的气温（重建高维数据）。

“秩”（Rank）就是这个矩阵中“独一无二”、“无法被其他数据组合所替代”的基础信息的最小数量。秩越低，说明冗余度越高，可压缩性越强。

Lora原理

Thinking：什么是“内在的低秩特性”？

- ΔW是什么：当一个预训练好的大模型（比如ChatGPT）去学习一个新任务时，其内部神经元的连接权重（一个巨大的矩阵 W）需要做出细微的调整。这个调整量就是 ΔW（权重更新）。
- 为什么说它“内在”是低秩的？

尽管 ΔW 在形式上是一个巨大的矩阵（例如有1000x1000=100万个参数），但研究者发现，驱动模型学会新任务所需要的“有效变化”其实非常集中和简单。

- 数学体现：对实际微调产生的 ΔW 矩阵做SVD分解，会发现它的奇异值衰减得非常快。绝大部分奇异值都接近零，只有前面几个奇异值特别大。

=> 真正起作用的更新信息，都集中在那几个最大的奇异值所对应的方向上。其余的都是可以忽略的冗余信息。

=> 这是LoRA算法的核心假设。

矩阵分解与猜你喜欢

什么是矩阵分解

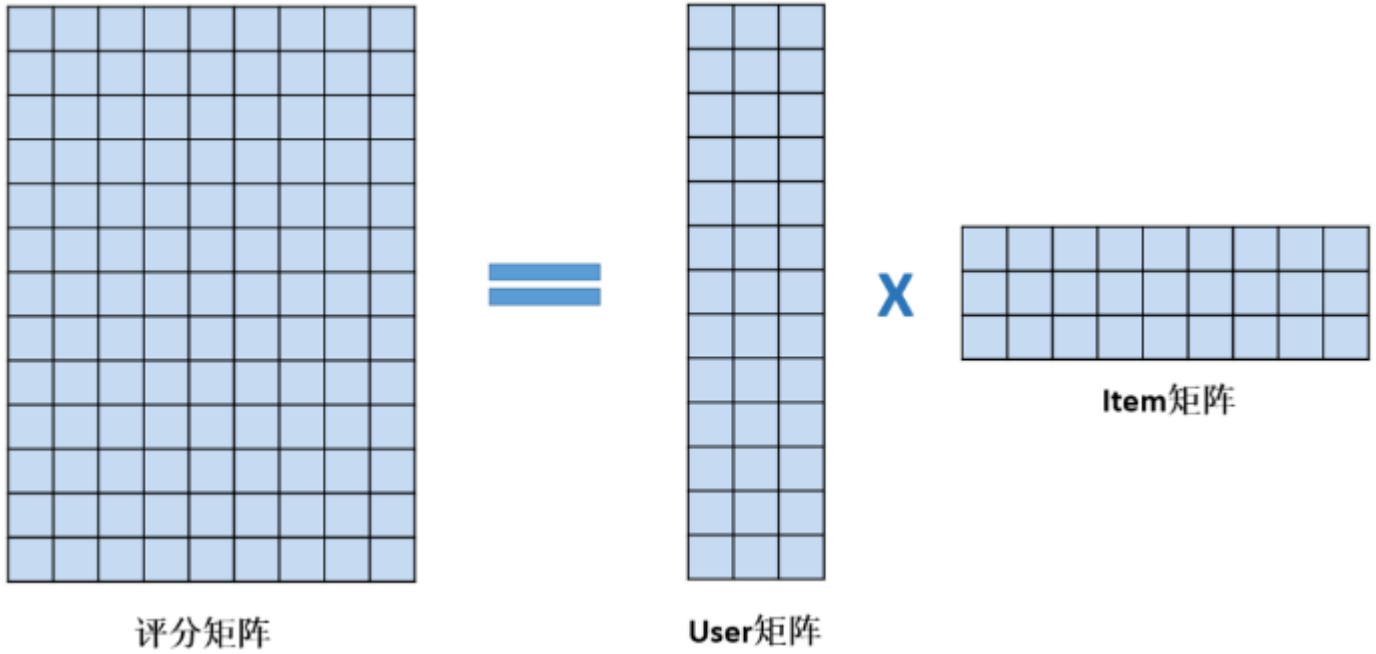
我们假设一个场景：为海量的用户和商品进行推荐：

- 为用户找到其感兴趣的item推荐给他
- 用矩阵表示收集到的用户行为数据，12个用户，9部电影

	红海行动	战狼2	湄公河行动2	汪汪队立大功	小猪佩奇	超级飞侠	小时代4	致青春	同桌的你
user 1	1	1							
user 2	1		1						
user 3		1	1						
user 4	1	1							
user 5				1	1				
user 6				1		1			
user 7					1	1			
user 8				1	1				
user 9				1	1				
user 10							1	1	
user 11								1	1
user 12							1		1

什么是矩阵分解

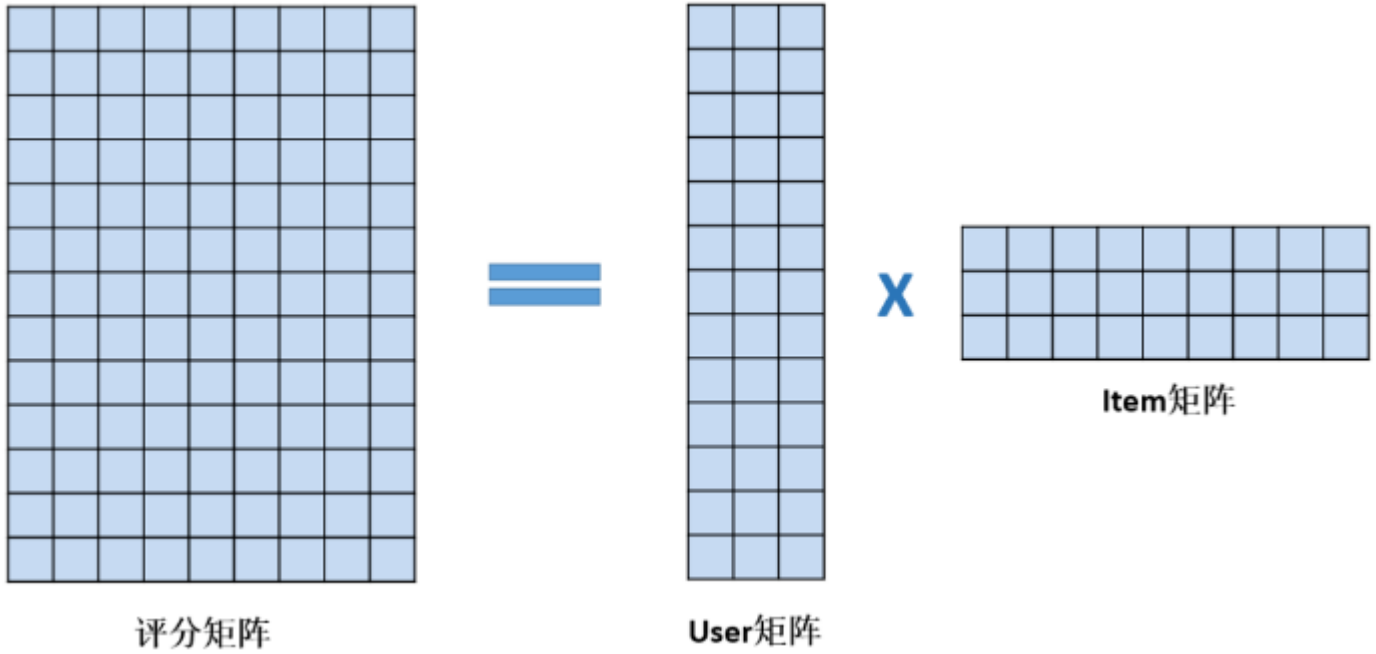
- 矩阵分解要做的是预测出矩阵中缺失的评分，使得预测评分能反映用户的喜欢程度
- 可以把预测评分最高的前K个电影推荐给用户了。



什么是矩阵分解

如何从评分矩阵中分解出User矩阵和Item矩阵

- 只有左侧的评分矩阵R是已知的
- User矩阵和Item矩阵是未知
- 学习出User矩阵和Item矩阵，使得User矩阵*Item矩阵与评分矩阵中已知的评分差异最小 => 最优化问题



什么是矩阵分解

- 观察User矩阵：用户的听歌爱好体现在User向量上
- 观察Item矩阵，电影的风格也会体现在Item向量上
- MF用user向量和item向量的内积去拟合评分矩阵中该user对该item的评分，内积的大小反映了user对item的喜欢程度。内积大匹配度高，内积小匹配度低。
- 隐含特征个数k，k越大，隐类别分得越细，计算量越大。

	动作	动画	爱情
user 1	0.93	-0.08	0.07
user 2	0.93	-0.21	0.07
user 3	0.9	-0.08	0.08
user 4	0.93	-0.08	0.08
user 5	0.11	0.81	-0.51
user 6	0.12	0.81	-0.51
user 7	0.11	0.74	-0.58
user 8	0.02	0.74	-0.51
user 9	-0.07	0.49	0.77
user 10	-0.07	0.49	0.77
user 11	-0.17	0.49	0.78
user 12	-0.17	0.49	0.78

X

	红海行动	战狼2	湄公河行动2	汪汪队立大功	小猪佩奇	超级飞侠	小时代4	致青春	同桌的你
动作	0.99	0.99	0.89	0.23	0.03	0.03	0.08	-0.11	-0.11
动画	0.09	-0.14	-0.14	0.87	0.58	0.58	0.23	0.36	0.36
爱情	0.16	0.07	0.07	-0.41	-0.47	-0.47	0.82	0.74	0.74

什么是矩阵分解

- 某个用户u对电影i的预测评分 = User向量和Item向量的内积
- 把这两个矩阵相乘，就能得到每个用户对每部电影的预测评分了，评分值越大，表示用户喜欢该电影的可能性越大，该电影就越值得推荐给用户。

	红海行动	战狼2	湄公河行动2	汪汪队立大功	小猪佩奇	超级飞侠	小时代4	致青春	同桌的你
user 1	0.92	0.94	0.84	0.12	-0.05	-0.05	0.11	-0.08	-0.08
user 2	0.91	0.96	0.86	0.00	-0.13	-0.13	0.08	-0.13	-0.13
user 3	0.90	0.91	0.82	0.10	-0.06	-0.06	0.12	-0.07	-0.07
user 4	0.93	0.94	0.84	0.11	-0.06	-0.06	0.12	-0.07	-0.07
user 5	0.10	-0.04	-0.05	0.94	0.71	0.71	-0.22	-0.10	-0.10
user 6	0.11	-0.03	-0.04	0.94	0.71	0.71	-0.22	-0.10	-0.10
user 7	0.08	-0.04	-0.05	0.91	0.71	0.71	-0.30	-0.17	-0.17
user 8	0.05	-0.12	-0.12	0.86	0.67	0.67	-0.25	-0.11	-0.11
user 9	0.10	-0.08	-0.07	0.09	-0.08	-0.08	0.74	0.75	0.75
user 10	0.10	-0.08	-0.08	0.10	-0.08	-0.08	0.74	0.75	0.75
user 11	0.00	-0.18	-0.17	0.07	-0.09	-0.09	0.74	0.77	0.77
user 12	0.00	-0.18	-0.17	0.07	-0.09	-0.09	0.74	0.77	0.77

矩阵分解的目标函数

r_{ui} 表示用户u 对item i 的评分

当 >0时，表示有评分，当=0时，表示没有评分，

x_u 表示用户u 的向量，k维列向量

y_i 表示item i 的向量，k维列向量

用户矩阵X，用户数为N

$$X = [x_1, x_2, \ldots, x_N]$$

商品矩阵Y，商品数为M

$$Y = [y_1, y_2, \ldots, y_M]$$

用户向量与物品向量的内积，表示用户u 对物品i 的预测评分

矩阵的目标函数：

$$\min_{X,Y} \sum_{r_{ui} \neq 0} \left(r_{ui} - x_u^T y_i\right)^2 + \lambda \left(\sum_u \|x_u\|_2^2 + \sum_i \|y_i\|_2^2 \right)$$

实际评分

L2正则项，保证数值计算稳定性，防止过拟合

矩阵分解的目标函数

目标函数最优化问题的工程解法：

- ALS，Alternating Least Squares，交替最小二乘法
- SGD，Stochastic Gradient Descent，随机梯度下降

ALS求解方法

ALS, Alternative Least Square, ALS，交替最小二乘法

- Step1，固定Y 优化X
- Step2，固定X 优化Y
- 重复Step1和2，直到X 和Y 收敛。每次固定一个矩阵，优化另一个矩阵，都是最小二乘问题

ALS求解方法

ALS, Alternative Least Square, ALS，交替最小二乘法

- Step1，固定Y优化X

$$\min_{X_u} \sum_{r_{ui}\neq 0} \left(r_{ui} - x_u^T y_i\right)^2 + \lambda \sum_u \|x_u\|_2^2$$

将目标函数转化为矩阵表达形式

$$J(x_u) = \left(R_u - Y_u^T x_u\right)^T \left(R_u - Y_u^T x_u\right) + \lambda x_u^T x_u$$

$$R_u = [r_{ui_1}, \ldots, r_{ui_m}]^T \qquad Y_u = [y_{i_1}, \ldots, y_{i_m}]$$

用户u 对m个物品的评分　　m 个物品的向量

ALS求解方法

ALS, Alternative Least Square, ALS，交替最小二乘法

- 对目标函数J关于x_u求梯度，并令梯度为零，得

$$\frac{\partial J(x_u)}{\partial x_u} = -2Y_u\left(R_u - Y_u^T x_u\right) + 2\lambda x_u = 0$$

求解后可得：

$$x_u = \left(Y_u Y_u^T + \lambda I\right)^{-1} Y_u R_u$$

CASE：对Movelens进行电影推荐

数据集：MovieLens

下载地址：https://www.kaggle.com/jneupane12/movielens/download

主要使用的文件：ratings.csv

格式：userId, movieId, rating, timestamp

记录了用户在某个时间对某个movieId的打分情况

我们需要补全评分矩阵，然后对指定用户，比如userID为1-5进行预测

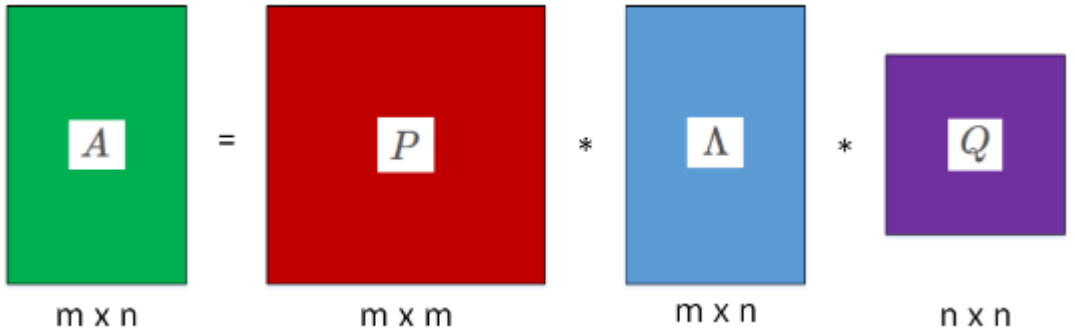
userId	movieId	rating	timestamp
1	2	3.5	1112486027
1	29	3.5	1112484676
1	32	3.5	1112484819
1	47	3.5	1112484727
1	50	3.5	1112484580
1	112	3.5	1094785740
1	151	4	1094785734
1	223	4	1112485573
1	253	4	1112484940
1	260	4	1112484826
1	293	4	1112484703
1	296	4	1112484767
1	318	4	1112484798
1	337	3.5	1094785709
1	367	3.5	1112485980
1	541	4	1112484603
1	589	3.5	1112485557
1	593	3.5	1112484661
1	653	3	1094785691
1	919	3.5	1094785621
1	924	3.5	1094785598
1	1009	3.5	1112486013
1	1036	4	1112485480
1	1079	4	1094785665

SVD矩阵分解

奇异值分解SVD

我们可以得到为奇异值分解

$$A = P\Lambda Q^T$$



P为左奇异矩阵，m*m维

Q为右奇异矩阵，n*n维

Λ对角线上的非零元素为特征值λ1, λ2, … , λk

在推荐系统中

左奇异矩阵：User矩阵

右奇异矩阵：Item矩阵

奇异值分解SVD

比如，$A=\begin{bmatrix}1 & 2\\ 1 & 1\\ 0 & 0\end{bmatrix}$

$AA^T=\begin{bmatrix}5 & 3 & 0\\ 3 & 2 & 0\\ 0 & 0 & 0\end{bmatrix}$

特征向量 $P=\begin{bmatrix}0.85065081 & -0.52573111 & 0\\ 0.52573111 & 0.85065081 & 0\\ 0 & 0 & 1\end{bmatrix}$

特征值 $\sigma_1=6.85410197,\sigma_2=0.14589803,\sigma_3=0$

$A^TA=\begin{bmatrix}2 & 3\\ 3 & 5\end{bmatrix}$

特征向量 $Q=\begin{bmatrix}-0.52573111 & -0.85065081\\ -0.85065081 & 0.52573111\end{bmatrix}$

特征值 $\sigma_1=6.85410197,\sigma_2=0.14589803$

代入求得：

$$\lambda_1=\sqrt{\sigma_1}=2.61803399 \qquad \lambda_2=\sqrt{\sigma_2}=0.38196601$$

$$P\Lambda Q^T=\begin{bmatrix}0.85065081 & -0.52573111 & 0\\ 0.52573111 & 0.85065081 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\\ 0 & 0\end{bmatrix}\begin{bmatrix}-0.52573111 & -0.85065081\\ -0.85065081 & 0.52573111\end{bmatrix}=\begin{bmatrix}1 & 2\\ 1 & 1\\ 0 & 0\end{bmatrix}$$

奇异值分解SVD

$$P\Lambda Q^T = \begin{bmatrix} 0.85065081 & -0.52573111 & 0 \\ 0.52573111 & 0.85065081 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} -0.52573111 & -0.85065081 \\ -0.85065081 & 0.52573111 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}$$

奇异值分解：

$$A = \lambda_1 p_1 q_1^T + \lambda_2 p_2 q_2^T + \cdots + \lambda_k p_k q_k^T$$

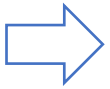
λ1为特征值，p1为左奇异矩阵的特征向量

q1为右奇异矩阵的特征向量

奇异值分解SVD

```
from scipy.linalg import svd

import numpy as np

from scipy.linalg import svd

A = np.array([[1,2],

              [1,1],

              [0,0]])

p,s,q = svd(A,full_matrices=False)

print('P=', p)

print('S=', s)

print('Q=', q)
```

$$P\Lambda Q^T = \begin{bmatrix} 0.85065081 & -0.52573111 & 0 \\ 0.52573111 & 0.85065081 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} -0.52573111 & -0.85065081 \\ -0.85065081 & 0.52573111 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}$$



```
P= [[-0.85065081 -0.52573111]
 [-0.52573111  0.85065081]
 [ 0.          0.        ]]
S= [2.61803399 0.38196601]
Q= [[-0.52573111 -0.85065081]
 [ 0.85065081 -0.52573111]]
[Finished in 0.6s]
```

奇异值分解SVD

如何理解SVD的作用

- 对于特征值矩阵，我们如果只包括某部分特征值，结果会怎样?

矩阵A：大小为1440*1080的图片

- Step1，将图片转换为矩阵
- Step2，对矩阵进行奇异值分解，得到p,s,q
- Step3，包括特征值矩阵中的K个最大特征值，其余特征值设置为0
- Step4，通过p,s',q得到新的矩阵A'，对比A'与A的差别

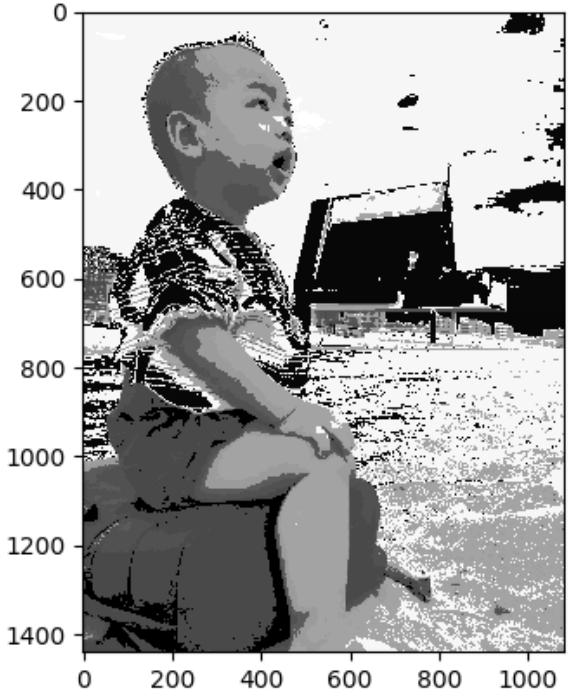
奇异值分解SVD

如何理解矩阵分解

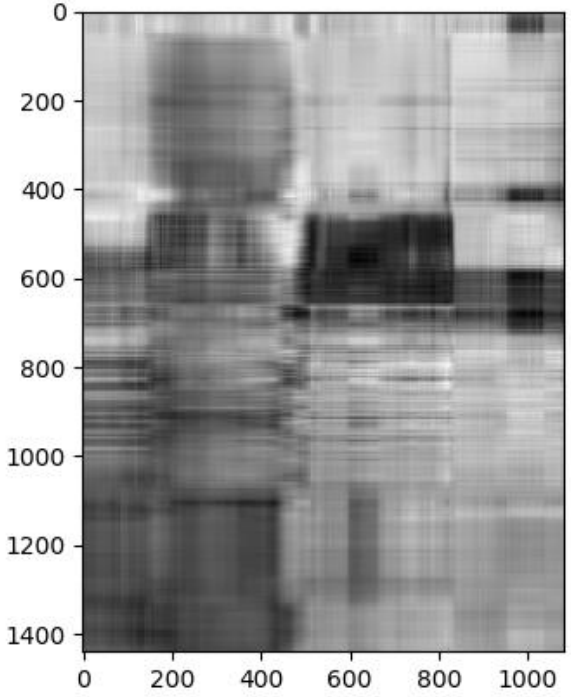
```
from scipy.linalg import svd
import matplotlib.pyplot as plt
# 取前k个特征，对图像进行还原
def get_image_feature(s, k):
        # 对于S，只保留前K个特征值
        s_temp = np.zeros(s.shape[0])
        s_temp[0:k] = s[0:k]
        s = s_temp * np.identity(s.shape[0])
        # 用新的s_temp，以及p,q重构A
        temp = np.dot(p,s)
        temp = np.dot(temp,q)
        plt.imshow(temp, cmap=plt.cm.gray, interpolation='nearest')
        plt.show()
# 加载256色图片
image = Image.open('./256.bmp')
A = np.array(image)
# 显示原图像
plt.imshow(A, cmap=plt.cm.gray, interpolation='nearest')
plt.show()
# 对图像矩阵A进行奇异值分解，得到p,s,q
p,s,q = svd(A, full_matrices=False)
# 取前k个特征，对图像进行还原
get_image_feature(s, 5)
get_image_feature(s, 50)
get_image_feature(s, 500)
```

奇异值分解SVD

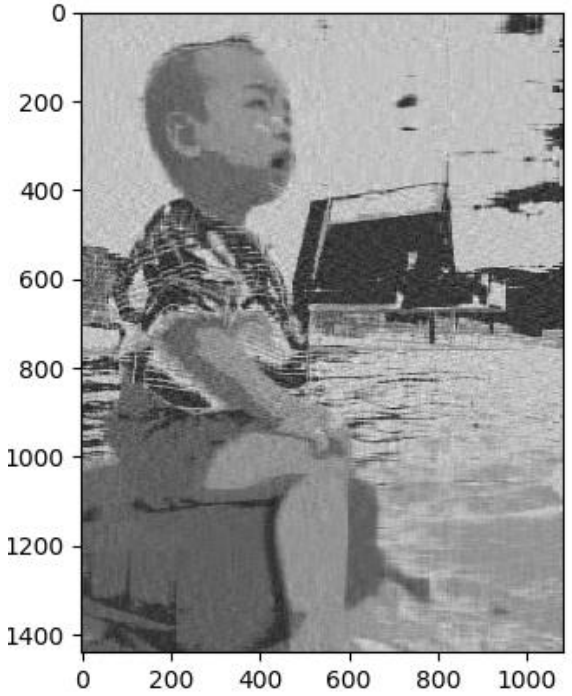
$$A = \lambda_1 p_1 q_1^T + \lambda_2 p_2 q_2^T + \cdots + \lambda_k p_k q_k^T$$



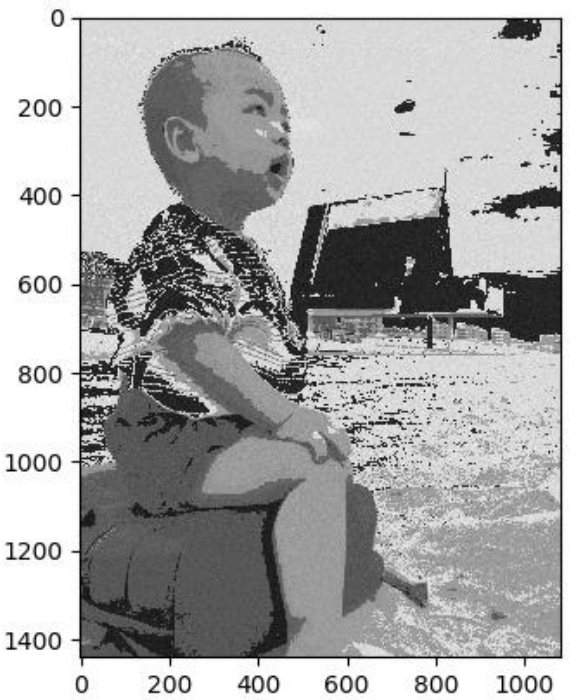
原始图片



k=5



k=50



k=500

少量的信息（比如10%），可以还原大部分图像信息（比如99%）

当K=50时，我们只需要保存 (1440+1+1080)*50=126050个元素，占比126050/(1440*1080)=8%

传统SVD在推荐系统中的应用

将user-item评分问题，转化为SVD矩阵分解

$$\begin{bmatrix} 1 & 5 & 0 & 5 & 4 \\ 5 & 4 & 4 & 3 & 2 \\ 0 & 4 & 0 & 0 & 5 \\ 4 & 4 & 1 & 4 & 0 \\ 0 & 4 & 3 & 5 & 0 \\ 2 & 4 & 3 & 5 & 3 \end{bmatrix} = \begin{bmatrix} -0.46 & 0.40 & 0.30 & -0.43 & 0.32 & -0.50 \\ -0.46 & -0.30 & -0.65 & 0.28 & 0.02 & -0.44 \\ -0.25 & 0.75 & -0.28 & 0.16 & -0.46 & 0.22 \\ -0.38 & -0.35 & -0.13 & -0.68 & -0.32 & 0.38 \\ -0.38 & -0.24 & 0.62 & 0.38 & -0.50 & -0.13 \\ -0.48 & -0.01 & 0.10 & 0.31 & 0.57 & 0.59 \end{bmatrix} \begin{bmatrix} 16.47 & 0 & 0 & 0 & 0 \\ 0 & 6.21 & 0 & 0 & 0 \\ 0 & 0 & 4.40 & 0 & 0 \\ 0 & 0 & 0 & 2.90 & 0 \\ 0 & 0 & 0 & 0 & 1.58 \end{bmatrix} \begin{bmatrix} -0.32 & -0.61 & -0.29 & -0.58 & -0.33 \\ -0.41 & 0.22 & -0.38 & -0.26 & 0.76 \\ -0.74 & 0.03 & -0.13 & 0.60 & -0.27 \\ -0.39 & -0.12 & 0.87 & -0.20 & 0.19 \\ 0.17 & -0.75 & -0.03 & 0.45 & 0.45 \end{bmatrix}$$

如果我们想要看user2对item3评分，可以得到

4=-0.46*16.47*(-0.29)+(-0.30)*6.21*(-0.38)+(-0.65)*4.40*(-0.13)+0.28*2.90*0.87+0.02*1.58*(-0.03)

- A中各元素=user行向量，item列向量，奇异值的加权内积

传统SVD在推荐系统中的应用

$$\begin{bmatrix} 1 & 5 & 0 & 5 & 4 \\ 5 & 4 & 4 & 3 & 2 \\ 0 & 4 & 0 & 0 & 5 \\ 4 & 4 & 1 & 4 & 0 \\ 0 & 4 & 3 & 5 & 0 \\ 2 & 4 & 3 & 5 & 3 \end{bmatrix} = \begin{bmatrix} -0.46 & 0.40 & 0.30 & -0.43 & 0.32 & -0.50 \\ -0.46 & -0.30 & -0.65 & 0.28 & 0.02 & -0.44 \\ -0.25 & 0.75 & -0.28 & 0.16 & -0.46 & 0.22 \\ -0.38 & -0.35 & -0.13 & -0.68 & -0.32 & 0.38 \\ -0.38 & -0.24 & 0.62 & 0.38 & -0.50 & -0.13 \\ -0.48 & -0.01 & 0.10 & 0.31 & 0.57 & 0.59 \end{bmatrix} \begin{bmatrix} 16.47 & 0 & 0 & 0 & 0 \\ 0 & 6.21 & 0 & 0 & 0 \\ 0 & 0 & 4.40 & 0 & 0 \\ 0 & 0 & 0 & 2.90 & 0 \\ 0 & 0 & 0 & 0 & 1.58 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -0.32 & -0.61 & -0.29 & -0.58 & -0.33 \\ -0.41 & 0.22 & -0.38 & -0.26 & 0.76 \\ -0.74 & 0.03 & -0.13 & 0.60 & -0.27 \\ -0.39 & -0.12 & 0.87 & -0.20 & 0.19 \\ 0.17 & -0.75 & -0.03 & 0.45 & 0.45 \end{bmatrix}$$

实际上，我们发现user矩阵的最后一列是没有用到的，而且我们还可以使用更少的特征，比如特征个数=2

得到近似解A'

$$\begin{bmatrix} -0.46 & 0.40 \\ -0.46 & -0.30 \\ -0.25 & 0.75 \\ -0.38 & -0.35 \\ -0.38 & -0.24 \\ -0.48 & -0.01 \end{bmatrix} \begin{bmatrix} 16.47 & 0 \\ 0 & 6.21 \end{bmatrix} \begin{bmatrix} -0.32 & -0.61 & -0.29 & -0.58 & -0.33 \\ -0.41 & 0.22 & -0.38 & -0.26 & 0.76 \end{bmatrix} = \begin{bmatrix} 1.41 & 5.18 & 1.27 & 3.73 & 4.37 \\ 3.20 & 4.22 & 2.92 & 4.85 & 1.05 \\ -0.61 & 3.55 & -0.57 & 1.16 & 4.91 \\ 2.89 & 3.39 & 2.64 & 4.18 & 0.45 \\ 2.59 & 3.45 & 2.37 & 3.95 & 0.89 \\ 2.52 & 4.77 & 2.29 & 4.51 & 2.54 \end{bmatrix}$$

Summary（Lora原理）

Lora (Low-Rank Adaptation of Large Language Models)

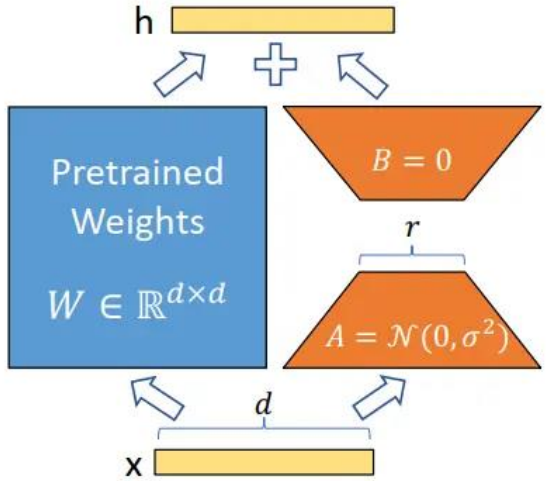
- 在原始预训练模型旁边增加一个旁路，通过低秩分解（先降维再升维）来模拟参数的更新量
- 将原模型 与 降维矩阵A，升维矩阵B分开

训练阶段，只训练B和A

推理阶段，将BA加到原参数，即h=Wx+BAx=(W+BA)x

不引入额外的推理延迟

- 初始化，A采用高斯分布初始化，B初始化为全0，这样训练开始时旁路为0矩阵
- 多任务切换，当前任务W0+B1A1，将lora部分减掉，换成B2A2，即可实现任务切换
- 秩的选取：对于一般的任务，rank=1,2,4,8即可，如果任务较大，可以选择更大的rank



Summary（Lora原理）

Lora (Low-Rank Adaptation of Large Language Models)

- 大模型的低秩适配器，即固定大模型，增加低秩分解的矩阵来适配下游任务
- Lora的优点：

1）一个中心模型服务多个下游任务，节省参数存储量

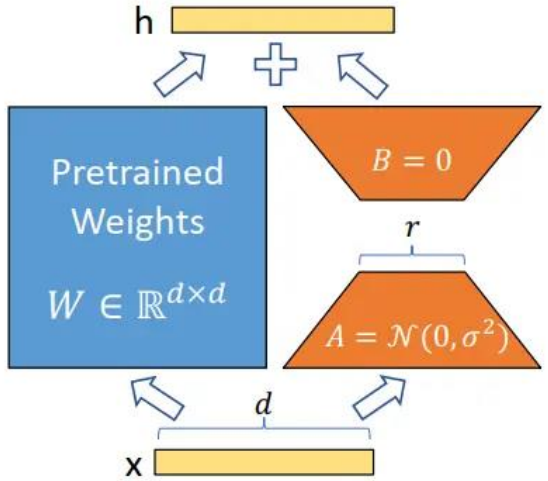
2）推理阶段不引入额外计算量

3）训练任务比较稳定，效果比较好

Thinking：如果不使用Lora进行训练会怎样？

如果训练集很小，比如20-30 samples，会把大模型带跑。

因为微调大模型会针对小样本进行过度拟合



PEFT库

Peft库：

https://github.com/huggingface/peft

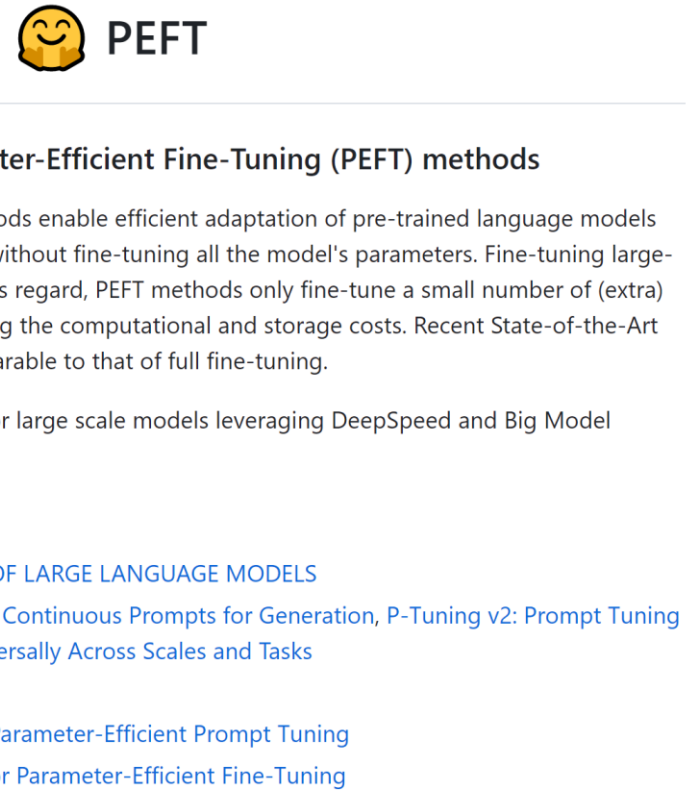
很方便地实现将普通的HF模型变成用于支持轻量级fine-tune的模型，目前支持4种策略：

1）LoRA:大模型的低秩适配器

2）Prefix Tuning: Optimizing Continuous Prompts for Generation

3）P-Tuning: GPT Understands, Too

4）Prompt Tuning: The Power of Scale for Parameter-Efficient Prompt Tuning



PEFT

State-of-the-art Parameter-Efficient Fine-Tuning (PEFT) methods

Parameter-Efficient Fine-Tuning (PEFT) methods enable efficient adaptation of pre-trained language models (PLMs) to various downstream applications without fine-tuning all the model's parameters. Fine-tuning large-scale PLMs is often prohibitively costly. In this regard, PEFT methods only fine-tune a small number of (extra) model parameters, thereby greatly decreasing the computational and storage costs. Recent State-of-the-Art PEFT techniques achieve performance comparable to that of full fine-tuning.

Seamlessly integrated with Accelerate for large scale models leveraging DeepSpeed and Big Model Inference.

Supported methods:

1. LoRA: LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS
2. Prefix Tuning: Prefix-Tuning: Optimizing Continuous Prompts for Generation, P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks
3. P-Tuning: GPT Understands, Too
4. Prompt Tuning: The Power of Scale for Parameter-Efficient Prompt Tuning
5. AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning

微调数据准备

微调数据准备

Thinking：LoRA既然能对参数高效微调，那么对数据的要求就会很低?

因为可训练的参数少，模型更加依赖我们提供的数据来精准学习 => 高质量的数据是微调成功的关键

高质量的数据：

- 一致性：提供的数据格式、指令风格和期望的输出需要统一。

比如：一条数据是“写一首诗” -> [诗歌]，下一条是“Summarize this: ...” -> [摘要]

=> 模型会感到困惑，不知道到底要学什么。

最佳实践：所有数据都应遵循相同的模板。比如

<指令>{instruction}</指令>

<输入>{input}</输入>

<回复>{response}</回复>

微调数据准备

- 准确性：

数据中的答案需要保证正确性，模型会学习数据中的所有 pattern

=> 包括里面的错误，garbage in garbage out

- 多样性：

在保证一致性的前提下，指令和输入要尽可能覆盖各种情况

=> 提高模型的泛化能力

微调数据准备

数据数量与模型尺寸、场景的关系

数据量没有绝对的标准，但它与模型大小和任务复杂度紧密相关。

模型规模 (参数量)	任务场景	建议数据量 (指令-回复对)	说明
~7B (70亿)	简单任务 (风格模仿、简单问答)	1000 - 5000 条	模型已有较强能力，LoRA微调主要是“引导”和“校准”。
~7B (70亿)	复杂任务 (推理、专业领域)	5000 - 50000+ 条	复杂逻辑和知识需要更多样本来教会模型。
~13B~70B	通用/复杂任务	1万 – 10万+ 条	模型容量更大，可以消化更多数据以学习更细微的模式
> 70B	继续预训练	海量数据 (GB级别文本)	目标是让模型学习语言本身而非指令遵循，需要大语料

Summary（微调数据准备）

Thinking：如何对模型微调进行数据准备？下

Step1：聚焦质量

花80%的时间在数据清洗、格式统一和答案校验上。1000条完美数据远胜于10万条杂乱数据。

Step2：评估数量

根据你的模型大小和任务难度，参考上述范围设定一个目标。从小规模开始（如1000条），进行实验，如果模型欠拟合（表现不好），再考虑增加数据。

任务越复杂，所需数据越多。

有一个“黄金起点”：对于许多任务，1000-6000条高质量指令数据已经可以产生很好的微调效果。

硬件需求与显存估算

硬件需求与显存估算

Thinking：微调显存估算的逻辑是什么？

微调时的总显存占用主要来自四个方面，可以用公式来估算：

总显存 ≈ (模型权重显存) + (优化器状态显存) + (梯度显存) + (前向传播激活值显存)

模型权重显存：这是最大的部分。模型通常以float16 (FP16)或bfloat16 (BF16)格式加载。

计算公式：模型参数量 (B) * 2字节

比如：一个7B（70亿）参数的模型，其权重显存约为 7 * 10^9 * 2字节 ≈ 14 GB。

硬件需求与显存估算

优化器状态显存：优化器（如Adam）为每个可训练参数保存的状态。

对于AdamW，它会为每个参数保存动量（momentum） 和方差（variance） 两个状态，通常也是FP16格式。

LoRA计算：假设LoRA可训练参数量为 L，则优化器状态显存约为 L * 4字节 * 2（两个状态）。

梯度显存：训练时反向传播计算的梯度。通常也是FP16格式。

LoRA计算：约为 L * 2字节（只存LoRA参数的梯度）。

前向传播激活值显存：计算过程中产生的中间变量（激活值），用于反向传播。这部分与批次大小（batch size） 和序列长度（sequence length） 强相关，估算复杂

=> 可以简单估算为模型权重显存的20%-50%。

硬件需求与显存估算

Thinking：如果微调一个7B（70亿）参数的模型需要多少显存?

LoRA的魔力在于，它通过大幅减少可训练参数量 L，从而极大地削减了第2、3部分的显存占用，使得第1部分（模型权重）成了绝对主力。

假设：

- 模型参数量：70B
- 使用AdamW优化器
- LoRA训练参数 L：设为1%，已经很充裕。
- 批次大小（batch size）：1
- 序列长度（sequence length）：512

显存估算：

- 模型权重 (FP16)：7e9 * 2字节 = ~14 GB
- 优化器状态：L * 4字节 * 2 = 7e7 * 8字节 ≈ 0.56 GB
- 梯度：L * 2字节 = 7e7 * 2字节 ≈ 0.14 GB
- 激活值 (估算)：14 GB * 0.3 ≈ 4.2 GB（按30%估算）
- 总显存估算：14 + 0.56 + 0.14 + 4.2 ≈ 19 GB

Summary（硬件需求与显存估算）

- 对于7B模型

一张24GB的消费级显卡（如RTX 4090） 可以流畅地微调，甚至可以使用稍大的batch size。

- 对于13B模型：

仅模型权重就需要26GB

=> 需要至少一张32GB的显卡（如RTX5090 32GB）或使用QLoRA技术。

- QLoRA是终极解决方案：

它将模型权重量化到4-bit，能将模型权重显存减少到约 模型参数量 * 0.5字节。

=> 微调7B模型仅需约6-8GB显存，让绝大多数显卡都能参与进来。

Summary（硬件需求与显存估算）

显存大头是模型本身，LoRA通过减少可训练参数来优化其余部分。

估算公式：总显存 ≈ (模型参数量 * 2字节) * (1 + 激活系数) + (L * 10字节)。激活系数通常取0.2~0.5。

硬件选择：

- 7B/8B模型：推荐 24GB 显存（如3090/4090）。
- 13B/14B模型：推荐 32GB+ 显存（如V100/A100），或使用QLoRA。
- 70B模型：必须使用QLoRA和多卡部署。

微调后的模型评估

微调后的模型评估

Thinking：我们投入了时间、数据和算力，如何证明微调后的模型变好了？

数据集划分——验证集与测试集

训练集：用于更新模型权重的数据 => 喂给LoRA微调的数据。

验证集：用于在训练过程中监控模型表现，调整超参数（如学习率），以及进行模型选择（比如选择训练得最好的那个checkpoint） => 它不能用于最终的性能报告。

测试集：用于最终、一次性的性能评估。它模拟了模型在“真实世界”中遇到的、从未见过的新数据上的表现。在整个微调过程中，测试集必须被严格“封存”，不能以任何形式用于训练。

测试集是模型的“期末考试”，绝对不能提前泄露考题。

微调后的模型评估

测试维度	测试内容	说明
1. 任务主指标	【核心对比】 使用任务相关的量化指标进行评估。 • 分类任务：准确率、F1分数、精确率、召回率 • 生成任务： BLEU, ROUGE (摘要), ChrF (翻译) • 问答任务： EM (精确匹配), F1	这是衡量微调是否成功的硬性标准。 直接反映了模型在目标任务上能力的提升幅度。
2. 通用能力保持	【关键检查】 测试模型在微调任务之外的通用能力是否退化。 • 常识推理（如：“西瓜的籽是什么颜色的？”） • 基础代码能力（如：写一个简单的排序函数） • 通用对话（如：“介绍一下你自己”）	防止模型出现“灾难性遗忘”。 确保模型没有因为学习新任务而“变傻”，破坏了原有的宝贵能力。
3. 泛化能力	测试模型在同一任务但不同表述或稍复杂场景下的表现。 • 使用与训练数据句式、词汇不同但意图相同的指令。 • 构造更复杂或包含干扰信息的输入。	检验模型是“死记硬背”了训练数据，还是真正“学会”了任务。 泛化能力差的模型无法在实际应用中落地。
4. 人工评估	【黄金标准】 由人类对模型生成的结果进行主观评分。 • 流畅度：输出是否通顺、符合语法? • 相关性：输出是否紧扣指令和输入? • 有用性：输出是否真正解决了问题或满足了需求?	很多生成任务（如对话、创意写作）的优劣难以用量化指标衡量。 人工评估能发现自动化指标无法捕捉的细微问题，是最终的质量关口。
5. 输出质量分析	【定性分析】 直接对比和分析模型在具体案例上的输入-输出。 • 正面案例：找出微调后效果提升显著的例子，展示成果。 • 失败案例：分析微调后仍表现不佳或出现负向变化的例子。	帮助直观地理解模型行为的变化，发现潜在问题。 为下一步迭代优化提供最直接的线索和方向。

Summary（微调后的模型评估）

- 准备阶段：提前从原始数据中划分出验证集和测试集。
- 训练中：使用验证集监控训练过程，防止过拟合（如果验证集指标开始下降，而训练集指标还在上升，说明模型可能过拟合了）。
- 训练后：

Step1：在测试集上运行基座模型和微调后的模型，记录所有主指标。

Step2：进行通用能力测试和泛化能力测试。

Step3：人工抽查测试集和通用能力测试中的一些样本输出，进行质量分析。

Step4：综合所有维度做出判断。主指标大幅提升，同时通用能力没有显著退化，才是一次成功的微调。


Thank You
Using data to solve problems