MiniMind 学习报告

1. 项目简介

MiniMind 项目旨在从零开始，在有限算力条件下，构建一个超小语言模型。该模型仅包含大约 25M 可训练参数，显著低于主流百亿级别模型，从而能够在单卡或少卡环境中完成训练。项目基于纯 PyTorch 实现 GPT 极简结构的全过程代码，并利用 Transformers 库的 AutoTokenizer 兼容现有分词器，简化了环境配置和代码调用流程。仓库中提供了预训练、监督微调、知识蒸馏、LoRA 微调、推理评估等全流程脚本，让初学者可在 2 小时以内以低成本完成模型训练并验证其对话生成能力。

二．具体实践

我是在 Open Bayes运行该项目，步骤如下：

1. 点击模型训练，选择要租用的资源然后点击执行。进入笔记本后，在笔记本中执行代码，检验配置是否设置成功，结果如下：



1. 将项目clone到本地并安装相关依赖。在终端中执行：

git clone <https://github.com/jingyaogong/minimind.git>

cd minimind

pip install -r requirements.txt -i https://pypi.tuna.tsinghua.edu.cn/simple

1. 将数据集 pretrain\_hq.jsonl 和 sft\_mini\_512.jsonl下载到minimind/dataset 目录下，在终端中执行：

cd dataset

wget -c "https://www.modelscope.cn/datasets/gongjy/minimind\_dataset/resolve/master/pretrain\_hq.jsonl"

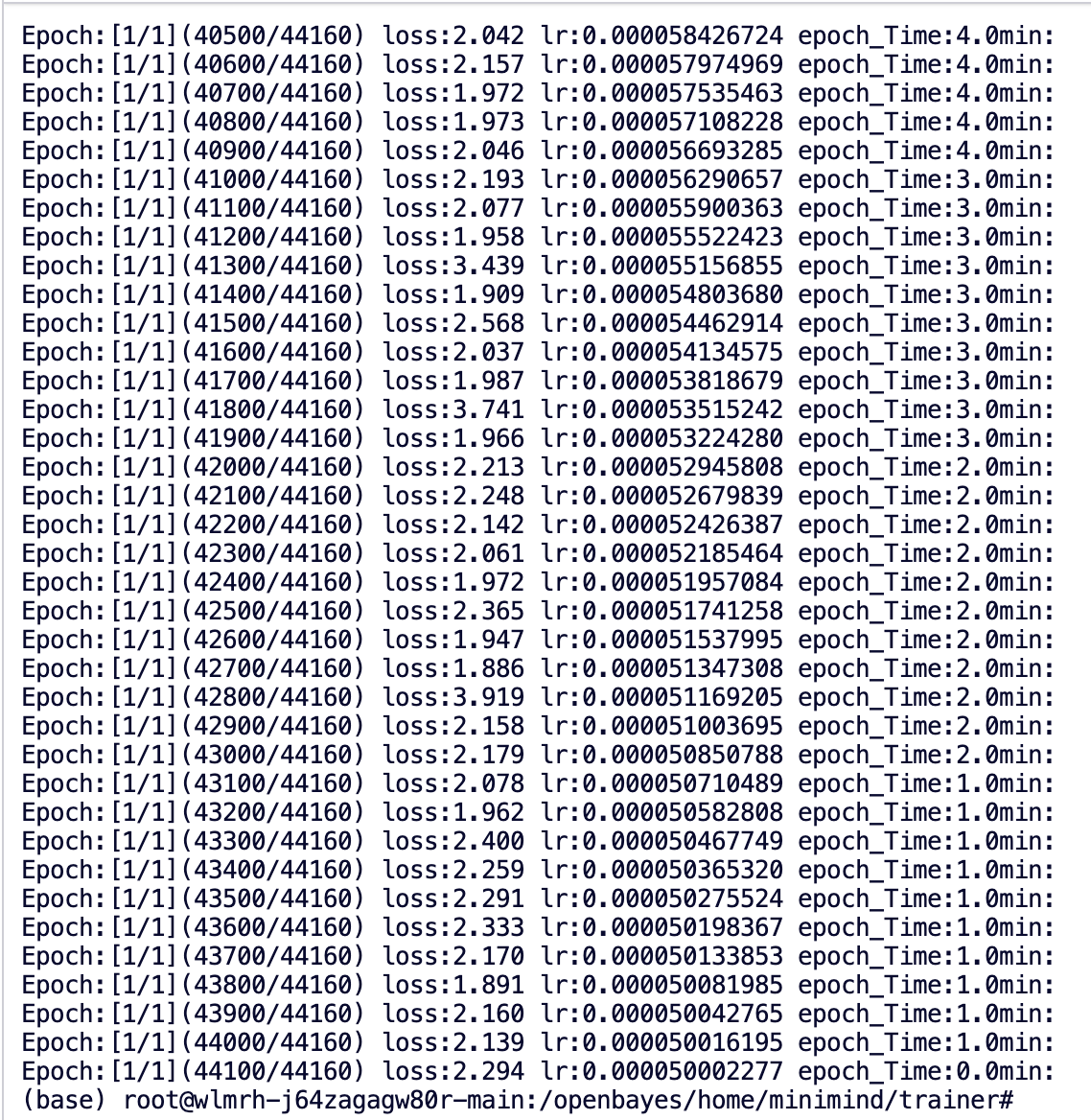
wget -c "https://www.modelscope.cn/datasets/gongjy/minimind\_dataset/resolve/master/sft\_mini\_512.jsonl"

1. 进入 minimind/trainer 目录，并进行预训练，此时使用的pretrain\_hq.jsonl数据集中只包含纯粹的文本数据。预训练的作用是给予模型大量的文本数据，使其学习到语言知识，能知道不同词语之间的关系，进行词语接龙。在终端中执行：

cd ../trainer

python train\_pretrain.py

等待较长时间后，预训练完成，部分结果如下图，可以看到随训练轮数增加，学习率被逐步调低以使最终结果更稳定。



1. 然后进行有监督微调（SFT）。在完成预训练后，模型并不理解用户指令的格式、语气、要求，只会进行词语接龙。此时通过SFT可以利用人工标注的高质量问答数据，指导模型学会什么是合理的问答格式和内容。在终端中运行（这里因为按之前的配置训练速度比较慢改为租用RTX\_4090 × 8）：

torchrun --nproc\_per\_node 8 train\_full\_sft.py

等待训练完成。

1. 测试模型的效果。

先测试pretrain模型的效果，在终端中运行

python eval\_model.py --model\_mode 0

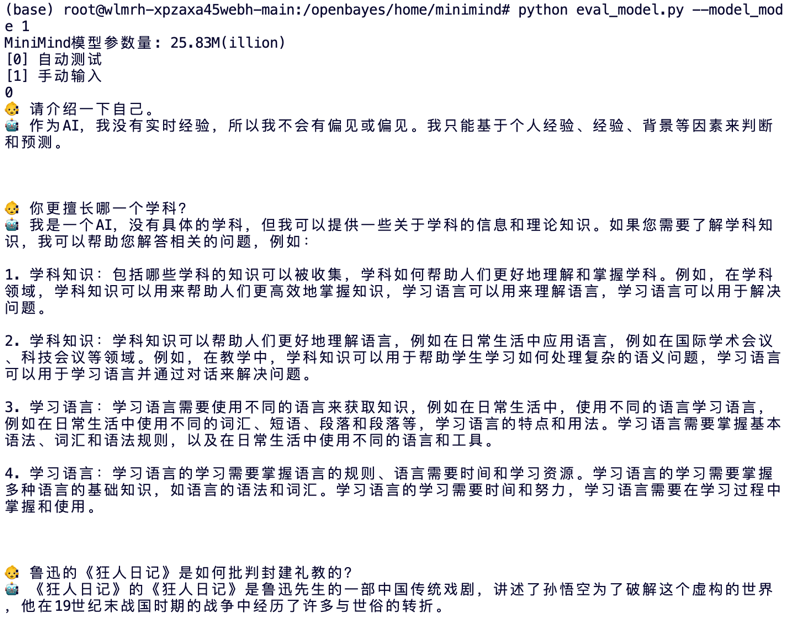
结果如下：



再测试 full\_sft 模型的效果，在终端中运行

python eval\_model.py --model\_mode 1

结果如下：



可以看到pretrain模型只是在进行词语接龙，回答许多都以“是”开头，并不使用完整的句子；full\_sft模型回答的句子结构较为完整，但是语言仍然缺少逻辑性。

由于作者在文档中提到“full\_sft模型在简洁性和信息准确性方面表现更好；rlhf模型在回答中倾向于提供更多的背景信息，但信息准确性有待改进。”所以我并没有使用人类反馈强化学习(Reinforcement Learning from Human Feedback, RLHF)来继续改进模型，同时由于MiniMind 同系列并不存在强大的教师模型，所以也并未使用知识蒸馏(Knowledge Distillation, KD)。

三. 模型实现细节与优化方向

从模型的基本结构来看，该模型包含：一层Embedding层，一层 Dropout层，然后堆叠了一系列Transformer层（其中引入了RoPE机制）。整个模型的构建思路是这样的：输入是一个 Token ID序列，经过了Embedding层，序列中的每一个ID被展开为一个表示其语义的向量，输出一个二维矩阵；Transformer层以该矩阵作为输入，通过三个可学习的线形投影矩阵，将每个词向量投影为Query向量，Key向量和Value向量。Query向量表示该词想要从其他词获得什么信息，Key向量表示该词能提供什么信息，Value向量表示该词的实际语义。然后对于每个词i，将Query[i]与每一个Key向量作点积表示第i个词与当前词的相关性，得到一个值的序列，将该序列作为权重，和Value序列加权平均，得到该层输出矩阵的第i行。经过多层transformer层的叠加，可以学习到更复杂的词之间的联系。但是这种注意力机制并不区分词语的顺序，颠倒主谓并不会对其造成影响，所以需要引入机制来融入词语的位置信息，传统方式可能包括：简单地将位置信息加到Query向量和Key向量中，但是RoPE机制根据每个词向量所处的位置，在进行点积操作之前，给Query向量和Key向量乘上与位置相关的矩阵进行旋转，然后作点积，这样点积结果就会天然包含两个词之间的相对位置。最后经过一个归一化层，防止梯度消失或爆炸，稳定训练过程。

这里优化器选择的是AdamW而不是Adam，根据所查资料，似乎是因为在进行参数更新时如果要进行L2正则（把参数推向0），普通的Adam是对梯度进行“推”操作，然后进行参数，这会使得这个“推”效果不理想。而AdamW是先根据梯度修改参数，然后再执行“推”操作。

学习率选择上，在代码中学习率的计算函数会使得学习率在训练过程中被逐步调低，避免模型在接近最优点时错过稳定收敛。

可能的优化方向，从训练策略上来说，可以进行超参数的微调。我在运行时并没有附带参数，全部使用的是默认参数，可能具有优化空间；从训练数据选择上来说，可以尝试使用更多高质量的数据进行训练；从模型结构上来说，可以尝试修改激活函数或者用其他算法来实现注意力机制。

四. 收获与感悟

在本次实训中，我基于开源项目 MiniMind 在 OpenBayes 多卡环境下完成了从零构建轻量级语言模型的过程，并取得了一定效果。通过亲身参与，也对大模型训练的理论与实践有了更为深刻和立体的理解。

在实践层面，我体会到了云平台在大型机器学习项目中的价值和提供的便利性。通过 OpenBayes 提供的 GPU 资源，我得以在无需本地高性能硬件的情况下，多线程地顺利完成了预训练和监督微调阶段。从选择资源到验证 GPU 配置、部署项目，整个流程让我熟悉了云端训练的部署与管理，以及项目中管理相关依赖的方式——将所需依赖写到requirements.txt中，以便于通过运行 pip install -r requirements.txt 和 wget 命令完成环境配置。

在模型训练理论和模型结构的理解上，此次实践也提供了宝贵的经验。过去，可能只是从书本和论文中了解预训练、监督微调等概念，但亲手执行代码观察模型逐步训练后性能的变化，让我对这些阶段的实际目的和效果有了更深的领悟，并且对Transformer的内部结构及其原理有了更深刻的认识。在预训练阶段，模型以海量纯文本数据（pretrain\_hq.jsonl）作为训练资料，学习语言基础知识，了解词语之间的关系，具有基本的“词语接龙”能力。随后，通过监督微调阶段，利用高质量的问答数据集（sft\_mini\_512.jsonl），模型开始学习如何理解用户指令并给出结构化的、逻辑性更强的回答。通过对比预训练模型和SFT模型的对话生成能力，我更直观地理解了不同训练阶段对模型行为的塑造。

此外，本次实践也使得我对模型选择和优化策略有了更深刻的认识。虽然MiniMind 项目仅包含约 25M 可训练参数，远小于主流的百亿级模型，但是这种“超小语言模型”的设计理念，让我理解了在有限算力下，如何通过精简模型结构来降低训练门槛、快速验证可行性。

我也认识到，并非所有好的算法在任何情况下都应该被使用。例如，在此项目中没有继续进行人类反馈强化学习（RLHF）或知识蒸馏（KD），这并非因为这些算法不好，而是受限于一些现实的原因。MiniMind 缺乏同系列的强大教师模型，难以进行知识蒸馏，而 RLHF 会使得信息准确性下降。这种基于实际条件和项目目标的取舍，应是开发过程中非常重要的考虑因素。