# 个人从零预训练1B LLM心路历程

原创 战士金 炼钢AI 2024年11月13日 21:34 北京

## 前言

项目开始于2024年3月初，当时朋友搞到了一台不知道能用多久的A100。这么棒的机器放着也是浪费，就琢磨着尝试从零训练一个小型号的LLM。其实在当时就有不少些这种“从零预训练LLM”的开源项目了，但是大多训练的数据量或者是模型都很小（几块4090+几十G数据就能跑起来），并没有暴露出一些工程上的问题，训练细节也没有分享的特别清晰。因此，我在制定训练LLM计划的时候有两个目标：

- 模型参数量和数据量不能特别的demo：参数量上B，数据量上T。
- 尽量详细的分享训练过程中的各种细节：让没有资源训练的同学能够了解到他们没有机会从实践得到的知识；让有训练资源的同学在复刻过程中少走弯路，以博客形式分享。

参考了TinyLlama项目的训练时间，估计了一下大概可以使用T级别的数据训练个1B大小的LLM（优先保证训练的数据量），耗时两个月左右。考虑到算力有限，决定这个LLM以中文语料为主（80%中文，20%英文），定位是中文LLM。

我给这个LLM命名为“Steel”(钢)，名称灵感来源于华北平原一只优秀的乐队“万能青年旅店（万青）”。乐队在做一专的时候条件有限，甚至在自己家里搭的录音棚，自称是在“土法炼钢”，但却做出了一张非常牛的国摇专辑。我们训练LLM的条件同样有限，但也希望能炼出好“钢”来。

Steel-LLM项目初期，我还打算做一件个人认为还比较有意思的事情：因为开源的LLM往往不会包含特别小众、个性化的知识内容，因此计划持续收集围观读者们的数据并训练到模型中，各种亚文化、冷知识、歌词、小众读物、个人的一些小秘密等等啥都行，通过这种方式让围观读者和Steel-LLM能建立起一些联系。但是收到的数据几乎没有，倒是有群友建议训练一些医疗数据进去，可能是他们公司的业务有需要。不过即使我在预训练里加了这部分数据，效果也是大概率比不过在qwen、llama这种大机构发布的模型基础上进行微调的。因此，项目后期我就把收集数据的这个事情从github主页上下架了，在我的第一篇文章中，读者还是能看到“收集数据”相关的描述。

笔者工作比较忙，项目过程中还遇到了算力断供的情况，因此断断续续进行了8个月的时间，过程中，也得到了 @lishu14 的帮助。得益于开源数据，最终Steel-LLM在ceval取得了38分，cmmlu上取得了33分的成绩，表现超过了一些大几倍的机构发布的早期LLM。

随着项目的推进，我其实已经分享过了如下4篇文章：
【从零训练Steel-LLM】预训练数据收集与处理
【从零训练Steel-LLM】预训练代码讲解、改进与测试
【从零训练Steel-LLM】模型设计
【从零训练Steel-LLM】微调探索与评估

因为项目周期比较长，内容也不少，我觉得有必要写下这篇汇总文章，读者对哪部分感兴趣可以再去翻当时更详细的细节。除此之外，**本文会侧重介绍在做各部分内容的时候笔者遇到的问题、回过头来的思考、以及群友/读者问到过的技术细节。**

github：

```
https://github.com/zhanshijinwat/Steel-LLM
```

模型地址：

```
https://hf-mirror.com/gqszhanshijin/Steel-LLM
https://modelscope.cn/models/zhanshijin/Steel-LLM
```

项目也有交流 裙，满200人了，加绿拉：a1843450905。

炼钢AI

个人公众号，首本RAG相关书籍《大模型RAG实战》、开源预训练项目Steel-LLM作者，...

7篇原创内容

公众号

## 数据收集与处理

Steel LLM使用的全部数据都是开源的，预训练阶段里Skywork/Skypile-150B数据集（600GB）、wanjuan1.0(nlp部分)（1TB）、starcoder的python、java、c++部分（200GB）占了绝大部分，英文数据比较少，只有wanjuan1.0有400GB。如果读者想现在也预训练训练一个，可以考虑使用MAP-NEO、BAAI的CCI3.0-HQ数据集等比较新的数据集。除了这些大数据量的预训练数据外，笔者还在预训练阶段加入了本应在SFT阶段加入的对话数据，比如百度百科问答数据、BELLE对话数据、moss项目对话数据等，这些对话数据仍然遵循如下这种对话形式的，数据的prompt部分在训练时也计算loss。

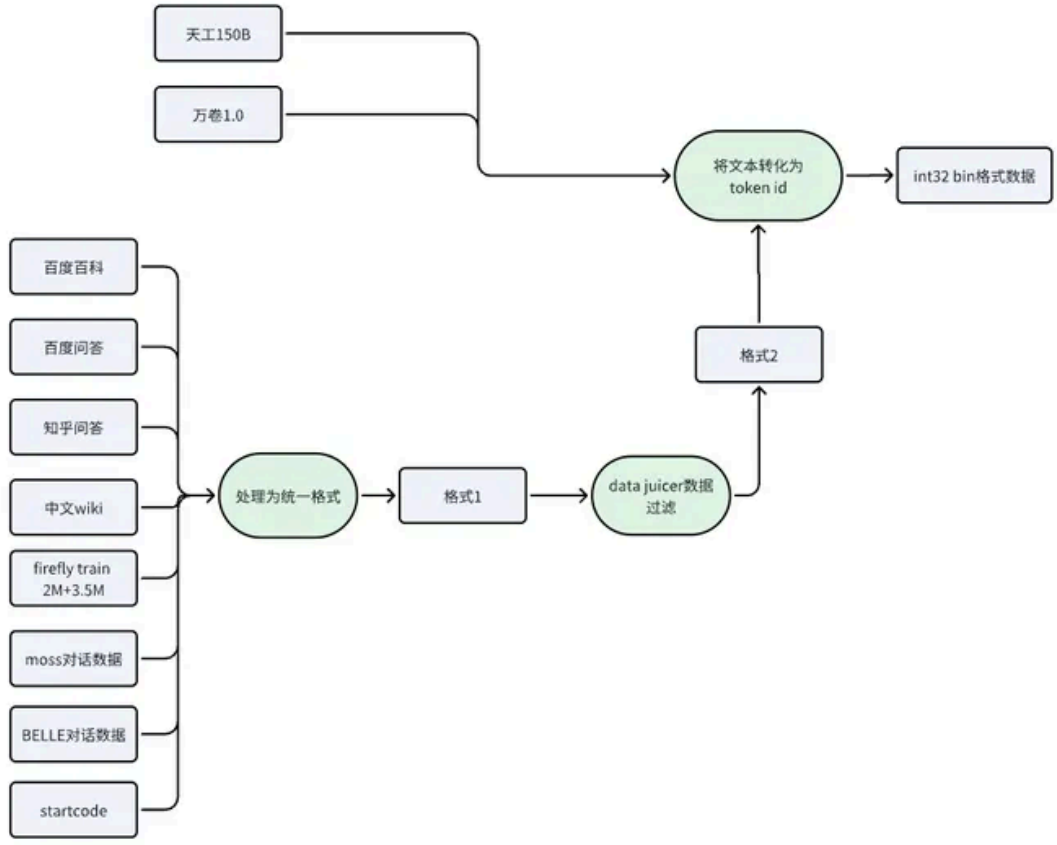
```
<|im_start|>system
You are a helpful assistant.<|im_end|>
<|im_start|>user
{问题}<|im_end|>
<|im_start|>assistant
{回答}<|im_end|>
```

笔者当时想的是：**如果我往预训练数据里加入对话数据，那么是不是不做SFT的模型也能有对话能力**？但实际预训练完的模型回答方式非常的四不像。。。对话和续写的能力都不是很理想（比如有时候以对话形式跟预训练后的Steel-LLM交互，他会进行续写），感兴趣的读者可以下载一个Steel LLM的预训练权重感受一下。

在预训练阶段加入SFT数据的方法在minicpm的训练过程中也使用了，但是只在**预训练末期**的退火阶段加入了SFT数据，这种方法预训练出来的模型指令跟随能力应该会比较强。如果预训练全程都杂糅SFT数据，这些少量的SFT数据会淹没在海量的原始文本中，起的作用会比较小。Steel-LLM的预训练没有退火阶段，退火阶段的的核心是“高质量”数据，但是高质量的定

义我觉得还是比较模糊的。不同类型的数据配比合理可以理解为是一种“高质量”，但是开源数据往往并没有给出数据的类型。经过各种规则筛选的数据也可以被称为高质量数据，但是开源数据基本都是经过数据清洗pipeline处理的。

Steel LLM对开源数据集的处理流程如下图所示。“处理为统一格式”和“将文本转化为token id”的代码在github仓库中给出，小一些的数据集使用了阿里的data-juicer工具进行了过滤，该工具将每个数据处理步骤抽象为一个算子，用户可以方便的配置yaml文件实现自定义的数据处理流程。但是有个问题是，data juicer处理数据的过程中产生的中间缓存有点多，大概占了几百GB的空间。Steel-LLM的数据处理pipeline并不是端到端的，比如下图中“格式1”和“格式2”的数据都要回落到硬盘上的，因此需要存同一份数据不同格式的多份拷贝，因此建议机器配有3~4T的硬盘空间。如果想在几个小时内处理完数据，需要开多进程，建议有100GB+的CPU内存。



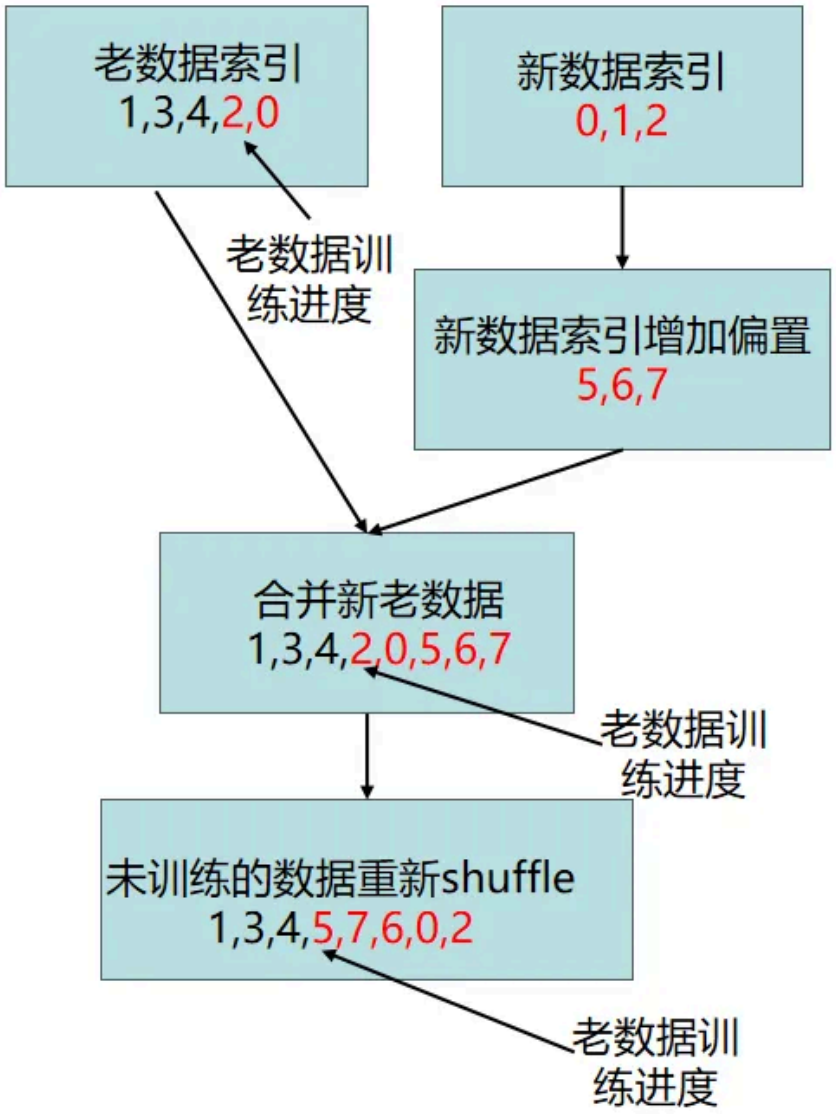

## 训练框架

训练代码这块主要借鉴了TinyLlama。有群友问我为啥不用Megatron ，最根本原因就是Steel LLM是个1B的模型，比较小，没有必要用Megatron。首先，Megatron支持pipeline并行、tensor并行，对训练大尺寸模型友好，但是在A100上，1B的模型使用原生pytorch的FSDP甚至是DDP就能稳定训练。其次，Megatron提供的算子融合、dataloader等能力，其他项目也有，而且简洁，更好集成和修改。

笔者也基于TinyLlama训练代码进行了如下几点改进：

- 兼容HuggingFace格式的模型：TinyLlama使用的模型结构是定义在项目里边的，如果使用TinyLlama项目训练其他模型不是很方便。笔者将其修改为使用HuggingFace的Transformers库的模式定义模型，开发者可以方便的更换自己想训练的模型结构。

- 数据训练进度恢复：预训练时间长，难免会出现中断的情况，想要继续训练的话，不仅要恢复模型和优化器的状态，还需要恢复数据训练的进度。TinyLlama main分支提供的恢复数据训练进度的方法比较简单粗暴，保存模型checkpoint时候也记录下迭代轮数，加载checkpoint之后直接跳过之前迭代过的轮数的数据，这种方式的的数据恢复要求预训练过程数据不能有变动。笔者对这块进行了进一步改进，保存所有数据的文件名、每条数据的索引，以实现数据进度的精准恢复。
- 支持训练过程中追加数据：预训练时间比较长，不免会有追加新数据的需求。基于第2点"数据训练进度恢复"改动，实现了新追加的数据索引会和老数据中未训练的数据索引（图中红色数字表示）重新shuffle的功能，防止加入新数据后，新老数据分布差异过大，影响后续的模型训练效果。具体原理如下图所示：




- 为了防止数据块被意外的重复追加到训练过程中，Steel-LLM还实现了使用hash值检测数据内容是否重复的功能，具体使用的是MD5哈希算法。对一个1.6g的文件进行哈希大约要9s，效率太低。因此选择只用一块数据的头部和尾部数据计算哈希值。

对于预训练来说，进行模型的训练效率的优化是必不可少的。卡少的情况下，即使预训练个小模型也得跑好几十天，那么训练效率优化个10%，就能省下好几天的电钱呢。个人或者小机构训练模型很难去优化底层的分布式训练机制，那么性价比最高的优化方式就是对模型的部分算子用算子融合。模型在训练时候，除了gpu的计算时间外，从显存把数据搬运到缓存也占用了很多时间。举个例子，算一个最简单的标准化(x-mean)/var，需要涉及到如下步骤：1.读x+读mean 2.写x-mean 3.读x-mean，读var 4.写(x-mean)/var，看起来十分的啰嗦，如果想让gpu端到端的计算结果（只读一次写一次），就需要做算子融合了，用cuda编

程或者triton编程都行。在LLM中，RMSNorm、RoPE、self-attention、交叉熵损失函数都是频繁读取数据的大户，因此能把这些操作融合掉能提高不少的训练时间。对self-attention进行算子融合的实现就是大名鼎鼎的flash attention。在我的往期文章中，笔者对各个算子的融合进行了消融，使用算子融合整体上能提升50%的训练效率，以及节省12%的显存，MFU（指模型一次前反向计算消耗的算力与机器能够提供的算力的比值）也能从43%提高到63%，大大提高了GPU的利用率。

分布式训练方式上，笔者沿用了FSDP（Fully-sharded data-parallel，完全分片数据并行）方法。有的读者对FSDP可能不是很熟悉，它的原理和deepspeed的ZeRO stage3一致，FSDP是pytorch的官方实现，将优化器参数、模型参数和梯度分布存储到不同的GPU上，节省分布式训练时候的显存，模型越大，显存节省的越多。

## 模型结构

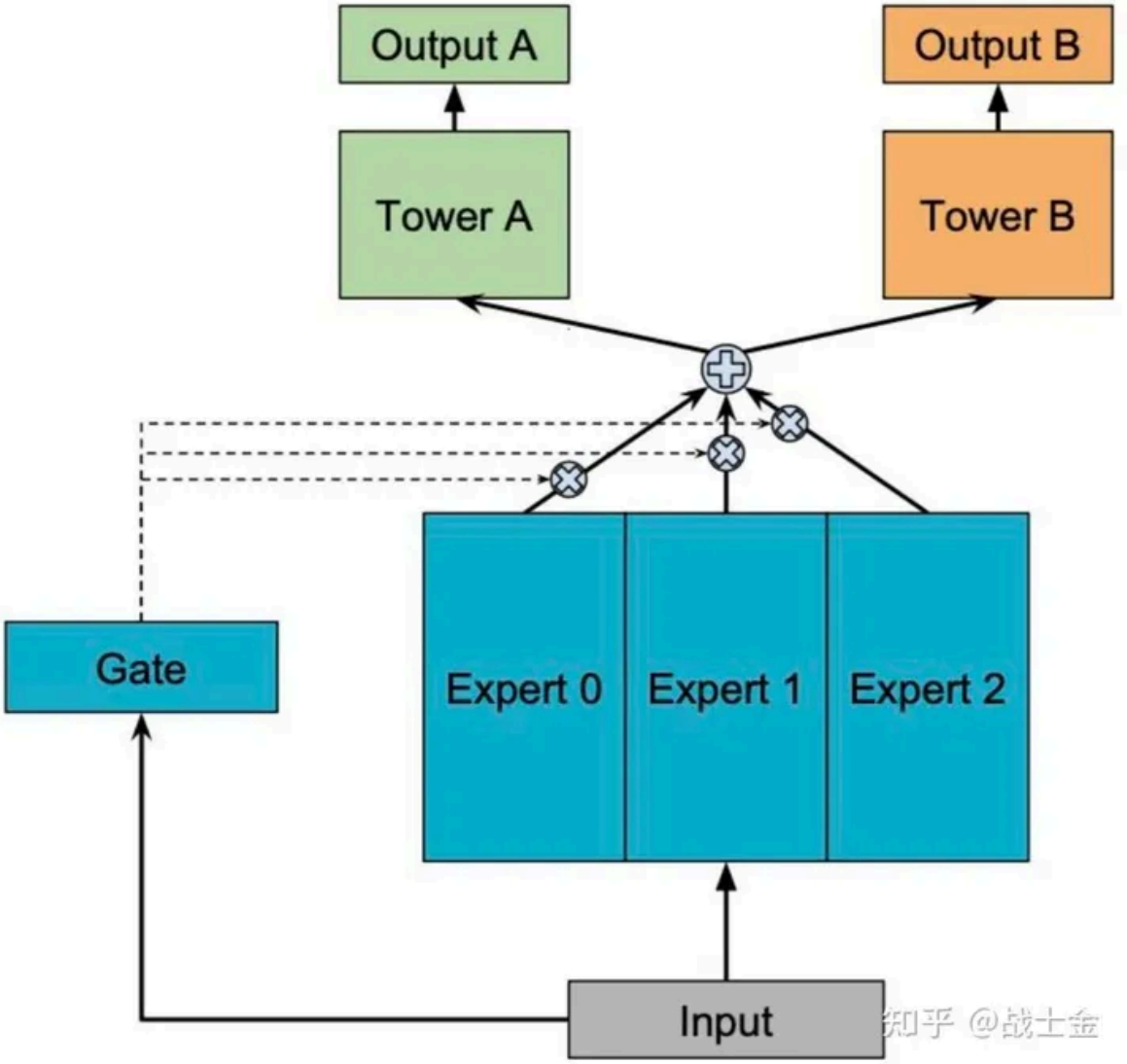
公司训练模型大部分都是沿用传统的LLM结构（self attention、RMSnorm、RoPE位置编码等），一来是LLM的效果主要是靠数据、二来是team leader也没必要去冒风险搞一些不一定有确定收益的创新。笔者训练Steel LLM没有任何的KPI压力，也愿意尝试进行一些结构上的修改，比较遗憾的是，笔者没有算力再训练一遍传统结构的LLM，去消融做的结构改动是否是好的。Steel LLM的tokenizer直接使用的是qwen的，并没有重新训练。

最开始应群友建议，我尝试了下训练recurrent gemma结构的模型，只不过它的pytorch实现训练效率太差了，和训transformer结构相比慢了几十倍，遂放弃，gemma是google训的模型，他们在TPU上训练时候做了不少的工程优化，训练效率才是可接受的。对于主流的LLM结构来说，self attention和FFN是两大核心模块。目前self attention这块的实现基本都用flash attention，修改self attention的实现比较困难。之前测试，如果不使用flash attention会掉25%左右的训练效率。因此，魔改结构的重点放到FFN上边。Steel LLM在FFN上的修改有两部分，soft MOE和SENet。

### （1）soft MOE

很多大型号的LLM使用hard MOE（每个token选择top k个FFN），目的是减少训练和推理时的计算量，但是hard MOE并不省显存，在训练和推理时仍然需要将完整的模型加载到显存中。考虑到只有1台机器训练Steel LLM，显存并不富裕，模型也比较小，想充分训练每一份参数，并且hard MOE并不好训练，需要考虑高效的token路由实现、训练稳定性以及负载均衡等问题。hard MOE存在种种问题，但笔者仍然想尝试一下MOE结构，因此使用了soft形式的MOE，这在搜广推多任务学习上被广泛使用。

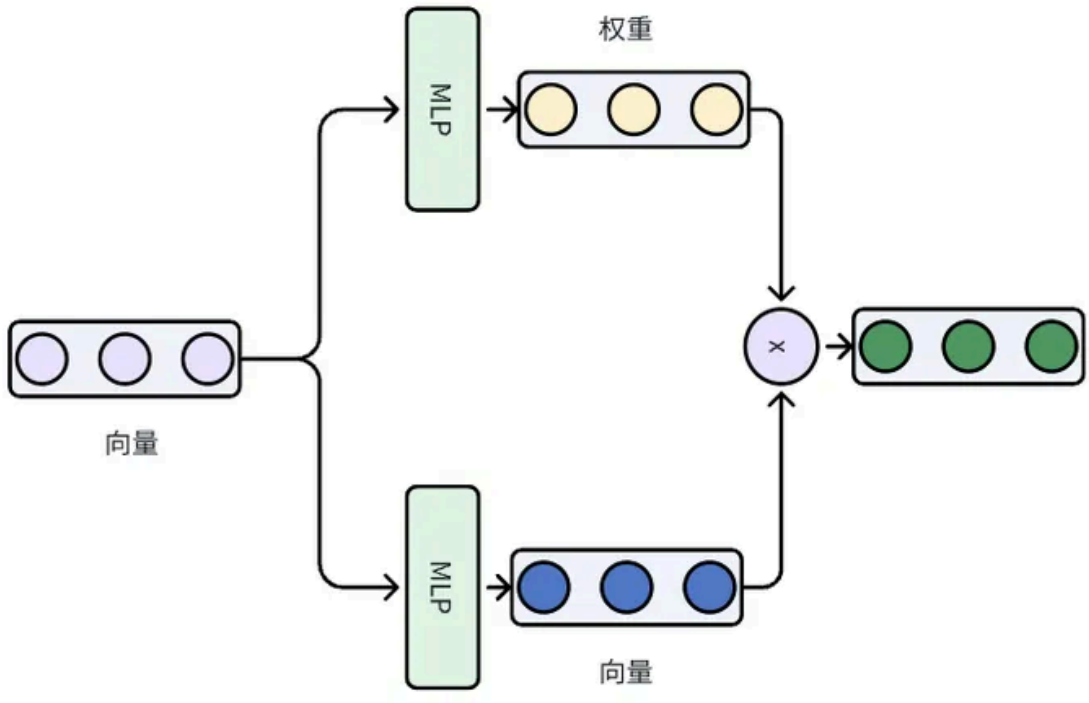
soft MOE的原理如下图所示，input就是一个向量，通过不同的专家网络计算出结果，然后再通过input计算出来的权重进行加权求和。在LLM中的FFN层使用soft MOE原理类似，只不过有多个input（即多个token），分别通过各个专家并进行加权求和。




笔者尝试了3版 soft MOE，第一版训练效率太差还费显存，第二版loss无法收敛到正常水平，最终尝试的第三版实现在效率和效果上才满足要求。这三版实现都在我的github仓库中。

### （2）SENet

SENet（Squeeze-and-Excitation Networks）来自于计算机视觉领域ImageNet 2017竞赛的冠军方案，目的是通过学习的方式来获取到每个图像特征通道的重要程度，通过重要程度加强或抑制特征通道，经过简单变换之后可变为如下图所示的形式：




我们再来看看Qwen模型的FFN层实现，大致可分为两层，它的第一层就是SENet的思想，用了gate_proj和up_proj，笔者将FFN的第二层也换成了SENet。

```
class Qwen2MoeMLP(nn.Module):
    def __init__(self, config, intermediate_size=None):
        super().__init__()
        self.config = config
        self.hidden_size = config.hidden_size
        self.intermediate_size = intermediate_size
        self.gate_proj = nn.Linear(self.hidden_size, self.intermediate_size,
        self.up_proj = nn.Linear(self.hidden_size, self.intermediate_size, bi
        self.down_proj = nn.Linear(self.intermediate_size, self.hidden_size,
        self.act_fn = ACT2FN[config.hidden_act]

    def forward(self, x):
        return self.down_proj(self.act_fn(self.gate_proj(x)) * self.up_proj(x
```

## 训练过程

预训练输入的最大序列长度是2048，将收集的数据训练了两个epoch，大概1.1T个token，需要训练1M个step。如果使用SXM版本的A100 80G*8，大概需要60天，H800*8需要30天左右。训练时的batchsize 为8，梯度累计步数为8。优化器使用的AdamW，最大学习率时3e-4，学习率衰减策略是CosineAnnealingWarmUp，这些都没啥特别的。训练的wandb链接如下：

```
https://api.wandb.ai/links/steel-llm-lab/vqf297nr
```

因为是单机8卡，训练还是比较稳定的，没有出现过啥error。唯一一次故障重启是机器网断了，wandb把训练进度给卡住了。

24年6月初，在Steel LLM训练了200k step的时候，朋友搞到的8卡A100被收回了。。。供应方说是临时拿走做下推理测试，但是之后再也没给回来。眼看Steel LLM就要成为烂尾项目，但非常的幸运，项目搁置一周多之后，某top3学校的老师说可以借我们1个多月的1台H800用，项目才得以完成，非常感谢这位老师。

## 微调和评估

24年8月中旬，Steel LLM预训练完成，打算开始开始做下后训练。目前先只考虑做sft，强化学习又是个大坑，工作有点卷，sft也是到了10月底才做完的，强化学习如果也要做一下估计得等到明年才能发布了（苦涩

微调阶段数据量不大，为了方便直接用llama factory训练了，选择如下4份数据：

- BAAI/Infinity-Instruct：今年8月发布的比较新的中英文sft数据，BAAI又是开源社区比较靠谱的机构，果断用起来。虽然有“做sft有少量高质量数据就行”的说法，但是我觉的小模型做sft还是数据多点好一些，毕竟预训练阶段学到内容相对较少，sft阶段补一补。Infinity-Instruct有700w条数据，量大管饱。
- wanjuan中文选择题部分：wanjuan中文选择题是预训练阶段就有的数据，因为微调之后的模型要在ceval等数据集上测试，因此sft阶段回炉重造一下，提高一下做题能力。
- ruozhiba：前段时间大火的数据集，从百度贴吧“弱智吧”扒的问题，用GPT4生成答案。
- 自我认知数据：希望让训练完的模型知道自己是Steel LLM，而不是回答自己是智能助手。模板来自于EmoLLM项目。

Steel LLM的预训练数据里80%以上都是中文的，所以只测了ceval和cmmlu这两个中文benchmark。第一版实验笔者用了700w条全量的Infinity-Instruct数据，ceval能有33%的准确率。后来发现Infinity-Instruct里90%的数据都是英文的，和预训练的数据分布严重不符（预训练数据里只有20%英文）。笔者之后从Infinity-Instruct里抽出了70w的中文数据，并糅合其他3个数据集，最终在ceval取得了38%的准确率，cmmlu取得了33%的准确率。

笔者还做了下刷榜测试，直接将cmmlu数据也放到sft数据里。在cmmlu上的正确率从33%提高了36%，在ceval上的准确率几乎没变化。这说明让模型去死记硬背答案也没那么容易。cmmlu在sft训练时候的标签只有一个选项字母，如果标签中有答案的解释，应该会效果更好一些。

除此之外，我还尝试了一下让模型以COT的方式进行答案生成，即先输出解释再输出答案，发现在benchmark上并没有拿到更好的结果。

Steel LLM模型和其他部分模型在ceval和cmmlu上的对比如下所示，其他模型的结果出自ceval、MiniCPM、MAP-Neo、CT-LLM的论文或技术报告。

| | CEVAL | CMMLU |
|---|---|---|
| Steel-LLM | 38.57 | 33.48 |
| Tiny-Llama-1.1B | 25.02 | 24.03 |
| Gemma-2b-it | 32.3 | 33.07 |
| Phi2(2B) | 23.37 | 24.18 |
| Deepseek-coder-1.3B-instruct | 28.33 | 27.75 |
| CT-LLM-SFT-2B | 41.54 | 41.48 |
| MiniCPM-2B-sft-fp32 | 49.14 | 51.0 |
| Qwen1.5-1.8B-Chat | 56.84 | 54.11 |
| ChatGLM-6B | 38.9 | - |
| Moss | 33.1 | - |
| LLAMA-65B | 34.7 | - |
| Qwen-7B | 58.96 | 60.35 |
| Gemma-7B | 42.57 | 44.20 |
| OLMo-7B | 35.18 | 35.55 |
| MAP-NEO-7B | 56.97 | 55.01 |

## 小结

随着开源数据的不断增多、算力价格不断降低，个人或小机构正经预训练一个小型号的LLM并不是遥不可及，希望Steel-LLM系列博客能对您在资源受限的情况下训练LLM产生一些启发或帮助。头一次做耗时这么长的个人项目，笔者精力和资源有限，确实存在一些没做到位的地方，比如：训练tokenizer、数据配比探究、全局数据清洗、模型英文能力较弱等。后续依算力情况，打算基于Steel LLM做一些微调方面的探索（样本筛选等）、强化学习或者是VLM，欢迎关注。

炼钢AI
个人公众号，首本RAG相关书籍《大模型RAG实战》、开源预训练项目Steel-LLM作者，...
7篇原创内容

公众号

从零预训练LLM 2