从token到patch，一种LLM加速训练策略

战士金 炼钢AI 2024年08月25日 23:02 北京

① 前言

此篇文章出自论文《Patch-Level Training for Large Language Models》，主要思路非常简单，就是把相邻的token embedding进行压缩聚合后输入到LLM中，进而缩短序列的长度加速训练，实验结果显示这种训练速度更快的训练方法，能比原始的LLM训练方法效果还要好，比较出乎预料。。。

```
论文链接：https://arxiv.org/abs/2407.12665
代码链接：https://github.com/shaochenze/PatchTrain/tree/main
```

② 方法

首先给下patch的定义，**将相邻的patch_size个token embedding取平均后的embedding，被称作patch**。seq_length长的token序列最终会转换为num_patches长的patch序列，代码如下。

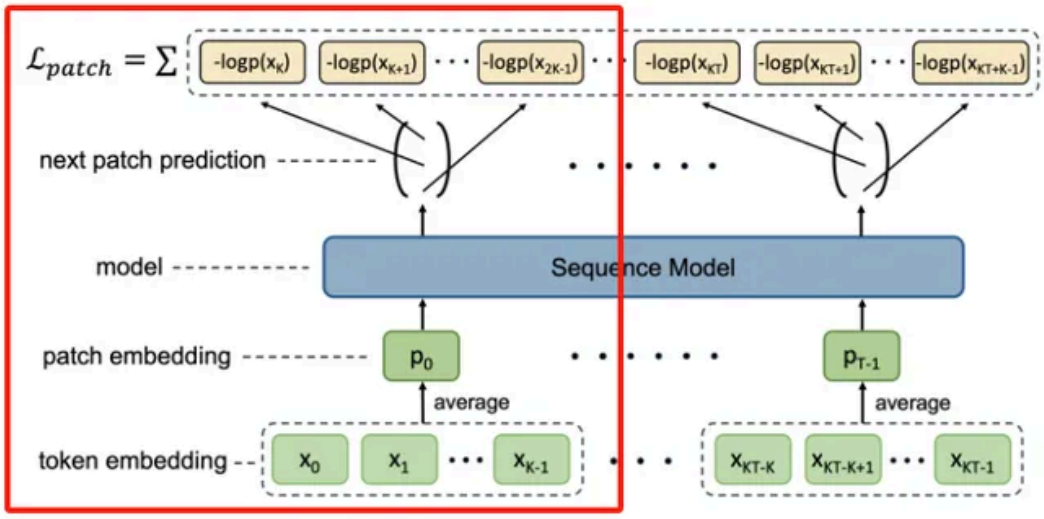
```
num_patches = seq_length // self.patch_size
inputs_embeds = inputs_embeds.view(batch_size, num_patches, self.patch_size,
```

训练分为两个阶段：
（1）将输入转换为patch粒度，并进行预测下一个patch训练
（2）加载第一阶段的模型参数，继续进行预测下一个token的训练
第一阶段更像是预训练任务的"预训练"阶段，学习patch之间的关系（有种patch里包含的token具有相同的注意力分值的感觉）。第二阶段恢复next token的训练，以对齐后边实际推理的情况。

这种两阶段的训练方式loss值（下图中橙色曲线）和常规从头就开始进行next token训练训练（下图中蓝色曲线）相比，甚至能得到更低的loss。假设我们用其中百分之x的数据进行第一阶段（patch级别）训练，每k个token聚合成1个patch，那么和从头就进行next token的训练相比，实际训练的数据量就会变为x/k+1-x。带入数据，当我们每4个token聚合为1个patch、2/3的数据进行patch级别的训练情况下，LLM实际计算的patch或token数量就会减小到一半。

Figure 2: Negative log-likelihood (NLL) loss on test set w.r.t the number of processed tokens during the training of 370M-parameter Transformers.

我们知道，常规LLM在训练时候，输入和输出都是token粒度的，因此可以通过直接预测下一个token的类别这种方式进行训练。但本文的方法中，当输入从token转为patch之后，标签仍然是token粒度的，或者说我们没办法构造patch粒度的标签。文中使用的的方式是，某个patch最终产生的logits和构成下一个patch的k个token的标签都计算交叉熵损失函数。示意图如下所示。



计算损失时的伪代码如下所示，logits形状是（B，L//patch_size，vocab_size），labels的形状是（B，L），L为转化为patch粒度之前的token的个数。

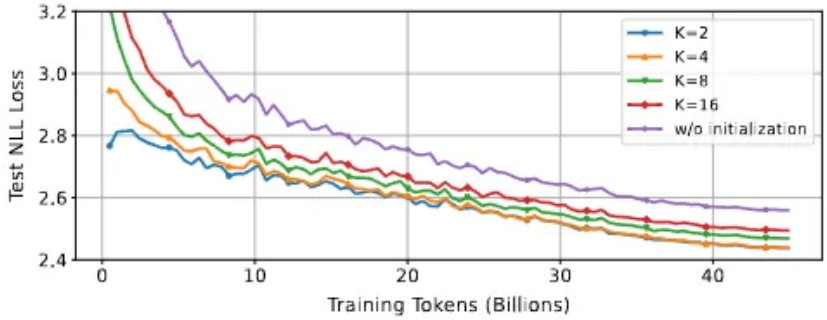
```
shift_logits = logits[..., :-1, :].reshape(-1, self.config.vocab_size)
shift_labels = labels[..., self.patch_size:].reshape(-1, self.patch_size)
loss_fct = CrossEntropyLoss()
loss = 0
for i in range(self.patch_size):
    loss = loss + loss_fct(shift_logits, shift_labels[:, i])
    loss = loss / self.patch_size
```

③ 实验结果

实验时使用了Pile数据集，包含360B个token。模型主干部分使用了传统的LLaMA结构。评测时既考察了PPL指标，也考察了在MMLU、HellaSwag等测试集上的准确率指标。不同尺寸模型下的实验结果如下所示。在370M模型参数量情况下，尝试了不同百分比（λ）的数据进行patch训练，可以看到用更多比例的数据进行patch阶段训练准确率是会降低的，这个是很符合直觉的，因为patch是token的聚合，本身就是有损的。但是当有2/3的数据进行patch训练的情况下，**各种尺寸的模型效果都要比从始至终在token粒度下训练的模型效果要好，这个就有点反直觉了。**有两个原因可能造成这种情况：（a）patch训练有更强的正则性质，减轻模型过拟合。（b）patch粒度的训练序列长度更短，模型能更容易学习捕捉不同位置token之间的关系。

Model Type	Cost	PPL	MMLU	HellaSwag	PIQA	WinoG	ARC-E	ARC-C	Average
Transformer-370M	1.0×	10.9	22.9	40.8	67.5	53.1	44.3	24.7	42.2
+ Patch ($\lambda = 1/2$)	0.625×	10.6	23.5	42.0	67.9	52.1	46.1	25.6	42.9
+ Patch ($\lambda = 2/3$)	0.5×	10.7	23.7	41.1	68.0	51.9	46.0	24.2	42.5
+ Patch ($\lambda = 4/5$)	0.4×	11.0	23.3	40.5	67.5	51.7	44.9	24.5	42.1
Transformer-780M	1.0×	9.2	24.4	48.5	69.0	55.4	49.0	26.7	45.5
+ Patch ($\lambda = 2/3$)	0.5×	9.1	24.1	49.1	70.6	54.8	51.3	28.2	46.3
Transformer-1.3B	1.0×	8.2	23.9	54.5	71.2	57.3	55.1	28.9	48.5
+ Patch ($\lambda = 2/3$)	0.5×	8.2	24.3	54.1	71.6	57.8	55.6	30.4	49.0
Transformer-2.7B	1.0×	7.1	25.3	62.2	74.3	61.5	61.2	34.3	53.1
+ Patch ($\lambda = 2/3$)	0.5×	7.2	25.4	61.9	74.9	62.4	61.9	34.6	53.5

作者对用多少个token（图中的K）聚合成一个patch进行了探究，如下图所示。K越大，loss越高，这其实比较符合直觉，K越大，聚合的token越多，信息损失越大。



作者也探究了在数据量恒定（左下图），和计算量恒定（右下图）的情况下，不同比例的数据进行patch粒度的训练（图中λ）的效果。可以看到PPL（越低越好）都是先下降后上升的。说明虽然patch粒度训练对模型是有益的，但是也需要留出足够的数据进行token级别的训练，因为最终测试时是在token粒度下的。

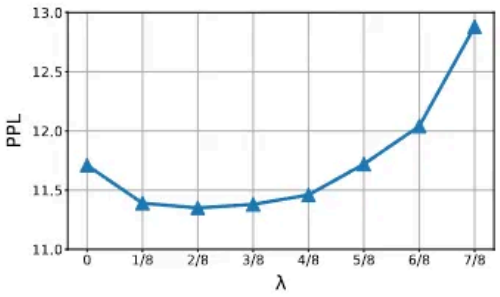


Figure 6: Effect of varying λ while keeping the data size constant.

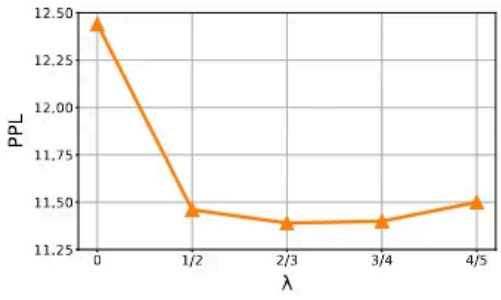


Figure 7: Effect of varying λ while keeping the computational cost constant.

感觉是个比较有意思的研究，不过应该不会有大厂真的用这种比较新颖的方法去训练吧。。。毕竟负责人不太会愿意承担训练效果不理想的风险。