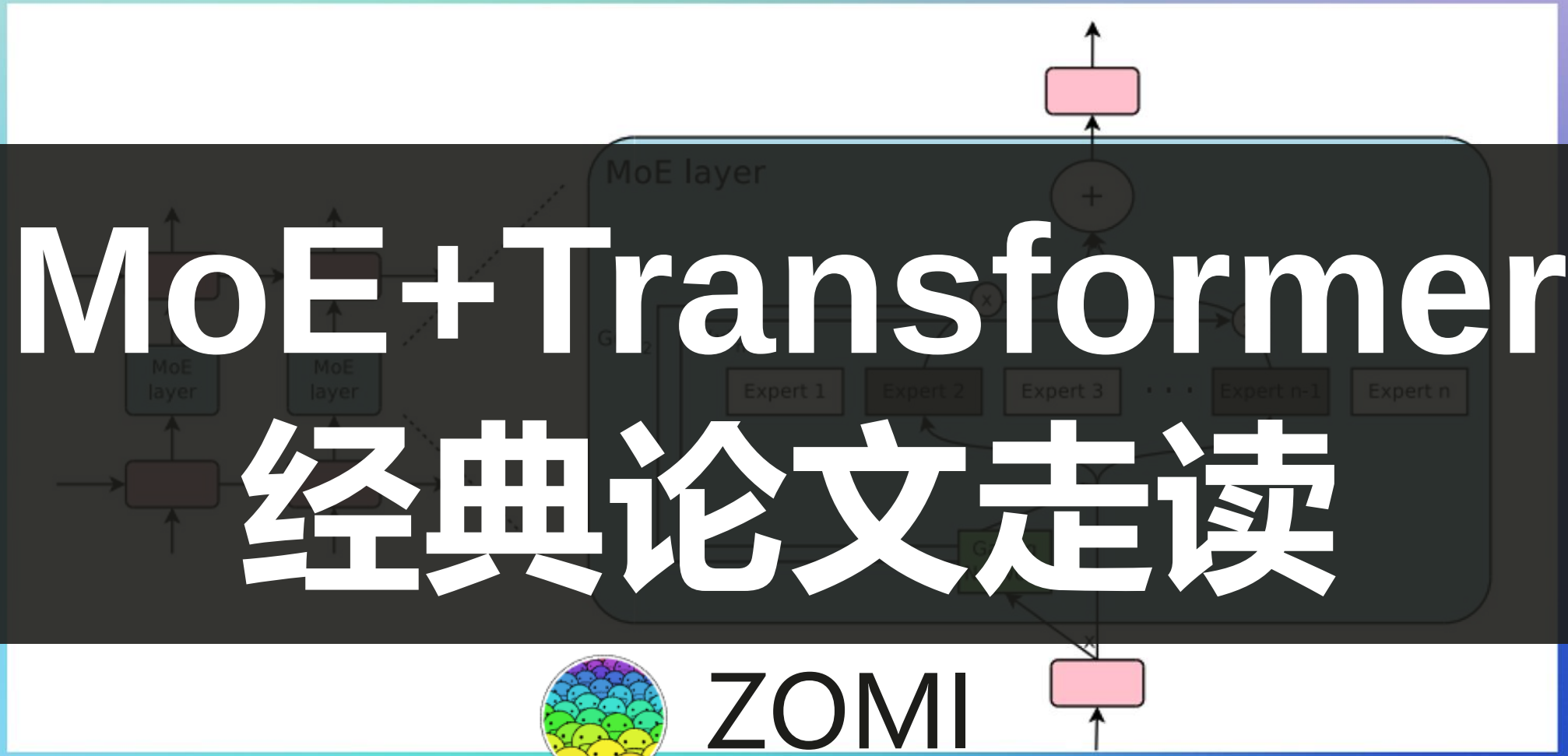
Mixture of Experts (MoE)



MoE+Transformer
经典论文走读

ZOMI

Contents

视频目录大纲

- MoE 混合专家
  - MoE 基本介绍 (4)
  - MoE 前世今生 (5)
  - MoE 核心工作
    - 奠基工作：Adaptive Mixtures of Local Experts
    - LSTM MoE：Outrageously Large Neural Networks ...
    - GShard: Scaling Giant Models with Conditional ...
    - Switch Transformers: Scaling to Trillion Parameter Models ...
    - GLaM: Efficient Scaling of Language Models with Mixture ...
    - ST-MoE: Designing Stable and Transferable Sparse Exp
  - MoE 架构原理 (3)
  - 大模型遇上 MoE (4)
  - 手撕 MoE 代码 (5)
  - MoE 的未来 (2)

03

GShard

基本介绍

- 2018年，随着Bert的发布，transformer结构彻底火了起来。2020年6月，Google发布《GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding》，把MoE用到了encoder-decoder结构的transformer模型上。MoE开始变成我们现在熟悉的样子了。
- GShard这个工作做了很多的实验，训了很多规模巨大的MoE模型，最大的达到了600B。训练的一系列模型的参数。

Id	Model	Experts Per-layer	Experts total	TPU v3 Cores	Enc+Dec layers	Weights
(1)	MoE(2048E, 36L)	2048	36684	2048	36	600B
(2)	MoE(2048E, 12L)	2048	12228	2048	12	200B
(3)	MoE(512E, 36L)	512	9216	512	36	150B
(4)	MoE(512E, 12L)	512	3072	512	12	50B
(5)	MoE(128E, 36L)	128	2304	128	36	37B
(6)	MoE(128E, 12L)	128	768	128	12	12.5B
*	MoE(2048E, 60L)	2048	61440	2048	60	1T

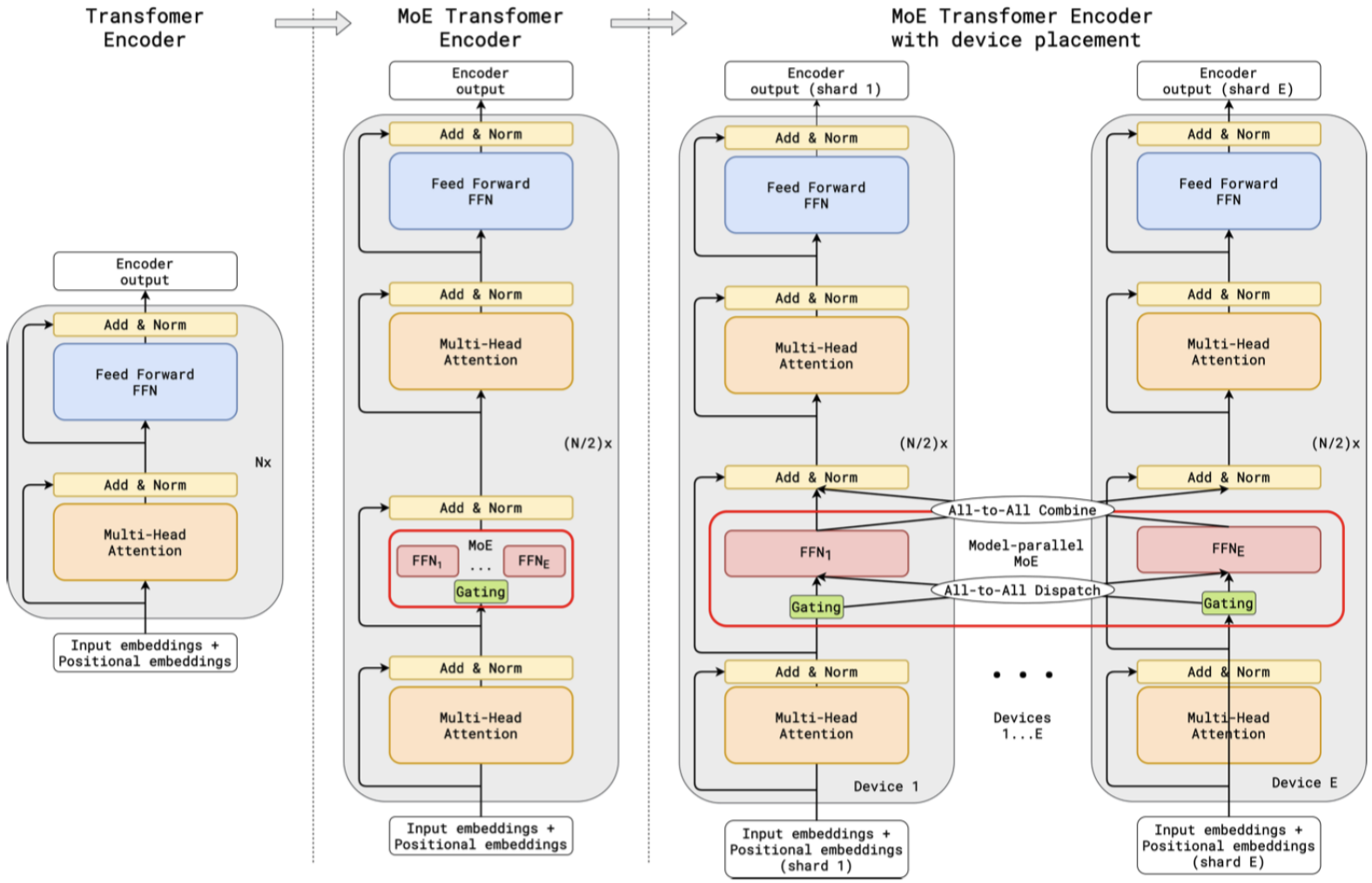
基本介绍

- 在expert数量的设计上，延续上面LSMT MoE工作的思路 —— expert越多，效果越好。
- GShard论文中很大的篇幅在介绍工程实现和优化，这也是MoE模型训练最大的痛点。关于工程框架的内容比较硬核，因此这里不会展开讲太多，而是关注在模型算法层面上。

Id	Model	Experts Per-layer	Experts total	TPU v3 Cores	Enc+Dec layers	Weights
(1)	MoE(2048E, 36L)	2048	36684	2048	36	600B
(2)	MoE(2048E, 12L)	2048	12228	2048	12	200B
(3)	MoE(512E, 36L)	512	9216	512	36	150B
(4)	MoE(512E, 12L)	512	3072	512	12	50B
(5)	MoE(128E, 36L)	128	2304	128	36	37B
(6)	MoE(128E, 12L)	128	768	128	12	12.5B
*	MoE(2048E, 60L)	2048	61440	2048	60	1T

模型设计

- Google在那段时间走的是encoder-decoder transfomer的技术路线，因此GShard也是基于encoder-decoder transfomer的模型结构。
- GShard的模型设计是，在encoder和decoder中，每两层把其中一个FFN层替换成MoE层。对于总共有N层的模型，则有N/2个MoE层



模型设计

- GShard在gating function的设计上提出了两个要求：
  - （1）负载均衡（2）高效扩展。
- 负载均衡和前面讲的一样，很好理解。而为什么要高效扩展，因为如果要对N个token分别进行E个expert的分配，在N能达到百万甚至千万级别，而E也有几百上千的情况下，就需要一个高效的分布式实现，以免其他计算资源等待gating function。

专家容量 expert capacity

- 为了确保负载平衡，我们不希望有少量expert需要处理很多token，因此强制规定了每一个expert所负责处理的token数量有一个最大值，这个最大值就叫专家容量，在这里设置为2N/E，相当于平均分配的量。
- 这个expert capacity通过GATE(·)给每个expert维护一个计数器来监控。如果一个token所选的两个专家当前处理量都已经超过设定的专家容量，那么这个token就不会被当前层的任何expert处理，而是直接通过残差链接透传到下一层。

分组分配 Local group dispatching

- 给所有输入token分成了G组，不同的组并行处理，每个组相应地也把组内专家容量变成2N/EG。
- 这样做相当于在前向推理时，把大的batch拆分成小的batch，每个小的batch就是一个group。这样做的好处是通讯的时候（特别是all2all）只需要在每个group内进行就可以了，减少了通讯量。
- 而进行反向计算的时候这些group可以合起来一起用，相当于进行了gradient accumulation。

辅助损失函数 Auxiliary loss

- 光设置专家容量并不能使得gating负载均衡，而且会导致大量溢出。参考前面LSTM MoE的工作，这里也定义了一个辅助损失函数，来帮助负载均衡。

随机路由 Random routing

- 前面提到，每层会选择最多top-2 expert来激活，就是因为有随机路由的机制。直观来说，就是认为如果top-1专家的权重很高，而第二个专家的权重如果较小，那很有可能只用第一个专家就足够解决问题了。
- 随机路由的机制是top-1的专家永远会被激活，而第二个专家如果权重很小，就认为它可以被忽略。具体来说，会以与第二个专家的权重g2成比例的概率激活第二个专家。

Thank you

把AI系统带入每个开发者、每个家庭、每个组织，构建万物互联的智能世界

Bring AI System to every person, home and organization for a fully connected, intelligent world.

ZOMI

GitHub https://github.com/chenzomi12/AIInfra

引用与参考

- https://mp.weixin.qq.com/s/6kzCMsJuavkZPG0YCKgeig
- https://www.zhihu.com/tardis/zm/art/677638939?source_id=1003
- https://huggingface.co/blog/zh/moe
- https://mp.weixin.qq.com/s/mOrAYo3qEACjSwcRPG7fWw
- https://mp.weixin.qq.com/s/x39hqf8xn1cUlnxEIM0_ww
- https://mp.weixin.qq.com/s/ZXjwnO103e-wXJGmmKi-Pw
- https://mp.weixin.qq.com/s/8Y281VYaLu5jHoAvQVvVJg
- https://blog.csdn.net/weixin_43013480/article/details/139301000
- https://developer.nvidia.com/zh-cn/blog/applying-mixture-of-experts-in-llm-architectures/
- https://www.zair.top/post/mixture-of-experts/
- https://my.oschina.net/IDP/blog/16513157

- PPT 开源：https://github.com/chenzomi12/AIInfra
- 夸克链接：https://pan.quark.cn/s/74fb24be8eff