

开关变压器：通过简单有效的稀疏性扩展到万亿参数模型

William Fedus*

LIAMFEDUS@GOOGLE.COM

Barret Zoph*

BARRETZOPH@GOOGLE.COM

Noam Shazeer

NOAM@GOOGLE.COM

Google, Mountain View, CA 94043, USA

编辑：亚历山大·克拉克

Abstract

在深度学习中，模型通常对所有输入重用相同的参数。专家混合（MoE）模型违背了这一点，而是为每个传入的示例选择不同的参数。结果是一个稀疏激活的模型——参数数量惊人——但计算成本恒定。然而，尽管教育部取得了一些显着的成功，但其广泛采用仍受到复杂性、通信成本和培训不稳定的阻碍。我们通过引入开关变压器来解决这些问题。我们简化了 MoE 路由算法，并设计了直观的改进模型，降低了通信和计算成本。我们提出的训练技术减轻了不稳定性，并且我们展示了大型稀疏模型可以首次使用较低精度 (bfloat16) 格式进行训练。我们基于 T5-Base 和 T5-Large (Raffel 等人, 2019) 设计模型，以在相同的计算资源下获得高达 7 倍的预训练速度提升。这些改进扩展到多语言设置，我们在所有 101 种语言中衡量 mT5-Base 版本的增益。最后，我们通过在“Colossal Clean Crawled Corpus”上预训练多达万亿参数的模型，推进了当前语言模型的规模，并实现了 T5-XXL 模型 4 倍的加速。12

关键词：专家混合、自然语言处理、稀疏性、大规模机器学习、分布式计算

A 。同等贡献。

1. Switch Transformer 的 JAX 代码和所有模型检查点可在 <https://github.com/google-research/t5x> 上获取
2. Switch Transformer 的 Tensorflow 代码可在 https://github.com/tensorflow/mesh/blob/master/mesh_tensorflow/transformer/moe.py 上获取

Contents

1 简介	
2 Switch Transformer	4
2.1 Simplifying Sparse Routing	5
2.2 Efficient Sparse Routing	6
2.3 Putting It All Together: The Switch Transformer	8
2.4 Improved Training and Fine-Tuning Techniques	8
3 Scaling Properties	11
3.1 Scaling Results on a Step-Basis	12
3.2 Scaling Results on a Time-Basis	13
3.3 Scaling Versus a Larger Dense Model	13
4 Downstream Results	14
4.1 Fine-Tuning	14
4.2 Distillation	16
4.3 Multilingual Learning	17
5 Designing Models with Data, Model, and Expert-Parallelism	18
5.1 Data Parallelism	20
5.2 Model Parallelism	20
5.3 Model and Data Parallelism	21
5.4 Expert and Data Parallelism	22
5.5 Expert, Model and Data Parallelism	22
5.6 Towards Trillion Parameter Models	22
6 相关工作	
7 讨论	
8 未来的工作	
9 结论	
注意力开关	
B 通过无令牌留守来防止令牌丢失	
C 鼓励专家探索	
较低计算范围中的 D 开关变压器	
E 上游与下游模型性能的关系	
F 开关变压器伪代码	

开关变压器

一、简介

大规模训练是实现灵活且强大的神经语言模型的有效途径 (Radford 等人, 2018; Kaplan 等人, 2020; Brown 等人, 2020)。简单的架构 (由大量的计算预算、数据集大小和参数数量支持) 超越了更复杂的算法 (Sutton, 2019)。Radford 等人采用了一种方法。

(2018); Raïl 等人。(2019); 布朗等人。(2020) 扩展了密集激活 Transformer 的模型大小 (Vaswani 等人, 2017)。虽然有效, 但计算量也极大 (Strubell 等人, 2019)。受到模型规模成功的启发, 但为了寻求更高的计算效率, 我们提出了一种稀疏激活的专家模型: Switch Transformer。在我们的例子中, 稀疏性来自于为每个传入示例激活神经网络权重的子集。

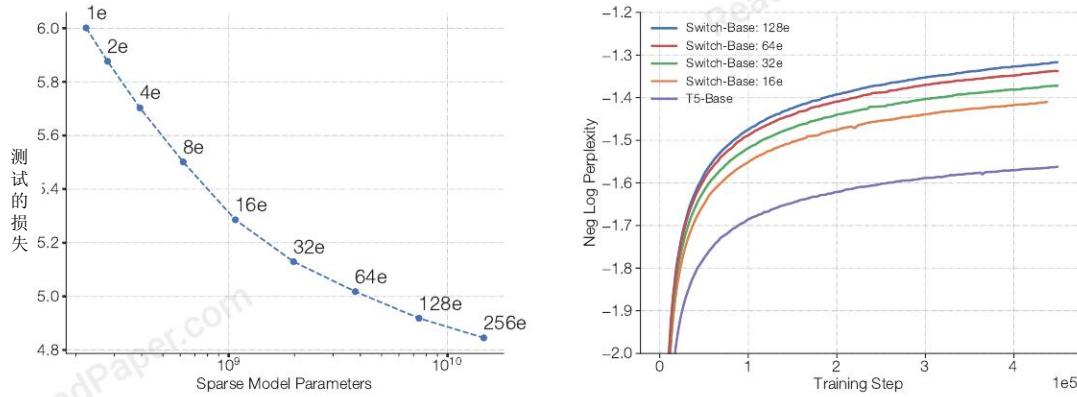


图 1: 开关变压器的缩放比例和采样效率。左图: 缩放道具—越来越稀疏 (更多专家) 的开关变压器的能力。右图: Switch Transformers 与 T5 的负对数困惑度 (Raïl 等人, 2019) 使用相同计算预算的模型。

稀疏训练是研究和工程的一个活跃领域 (Gray 等人, 2017 年; Gale 等人, 2020 年), 但截至目前, 机器学习库和硬件加速器仍然迎合密集矩阵乘法。为了拥有高效的稀疏算法, 我们从专家混合 (MoE) 范式开始 (Jacobs 等人, 1991; Jordan 和 Jacobs, 1994; Shazeer 等人, 2017), 并将其简化以产生训练稳定性和计算优势。MoE 模型在机器翻译方面取得了显着的成功 (Shazeer 等人, 2017 年、2018 年; Lepikhin 等人, 2020 年), 然而, 广泛采用受到复杂性、通信成本和培训不稳定的阻碍。

我们解决这些问题, 然后超越翻译, 发现这类算法在自然语言中具有广泛的价值。我们测量了多种自然语言任务以及 NLP 中三种机制的卓越扩展性: 预训练、微调和多任务训练。虽然这项工作侧重于规模, 但我们还表明 Switch Transformer 架构不仅在超级计算机领域表现出色, 而且在

即使只有几个计算核心也是有益的。此外，我们的大型稀疏模型可以被提炼 (Hinton et al., 2015) 为所有密集版本，同时保留 30% 的稀疏模型质量增益。我们的贡献如下：

- Switch Transformer 架构，它对 Mixture of Experts 进行了简化和改进。

缩放属性和针对强调优 T5 模型的基准 (Raïl 等人, 2019)，其中我们测量了 7 倍以上的预训练加速，同时仍然使用每个代币相同的 FLOPS。我们进一步证明，即使计算资源有限，仅使用两名专家，这种改进仍然有效。

成功地将稀疏的预训练和专门的微调模型提炼为所有密集模型。我们将模型大小减少了 99%，同时保留了大型稀疏教师 30% 的质量增益。

改进的预训练和微调技术：(1) 选择性精度训练，能够以较低的 bfloat16 精度进行训练 (2) 允许扩展到更多专家数量的初始化方案，以及 (3) 增加专家正则化可改善稀疏模型的微调和多任务训练。

衡量多语言数据上的预训练收益，我们发现所有 101 种语言都得到了普遍改进，其中 91% 的语言受益于 mT5 基线 4 倍以上的加速 (Xue 等人, 2020))。

通过有效地结合数据、模型和专家并行来创建具有多达一万亿个参数的模型，从而扩大神经语言模型的规模。这些模型将强调整的 T5-XXL 基线的预训练速度提高了 4 倍。

2、开关变压器

Switch Transformer 的指导设计原则是以简单且计算高效的方式最大化 Transformer 模型的参数数量 (Vaswani 等人, 2017)。Kaplan 等人对规模效益进行了详尽的研究。

(2020) 揭示了模型大小、数据集大小和计算预算的幂律缩放。重要的是，这项工作主张在相对所有数据量上训练大型模型作为计算上的最佳方法。

考虑到这些结果，我们研究了第四个轴：增加参数数量，同时保持每个示例的浮点操作 (FLOP) 不变。我们的假设是，参数计数与执行的总计算无关，是一个单独重要的缩放轴。我们通过设计一个稀疏激活模型来实现这一目标，该模型有效地使用专为密集矩阵乘法设计的硬件，例如 GPU 和 TPU。我们这里的工作重点是 TPU 架构，但此类模型可能在 GPU 集群上进行类似的训练。在我们的分布式训练设置中，我们的稀疏激活层在不同设备上分割独特的权重。因此，模型的权重随着设备数量的增加而增加，同时在每个设备上保持可管理的内存和计算占用空间。

开关变压器

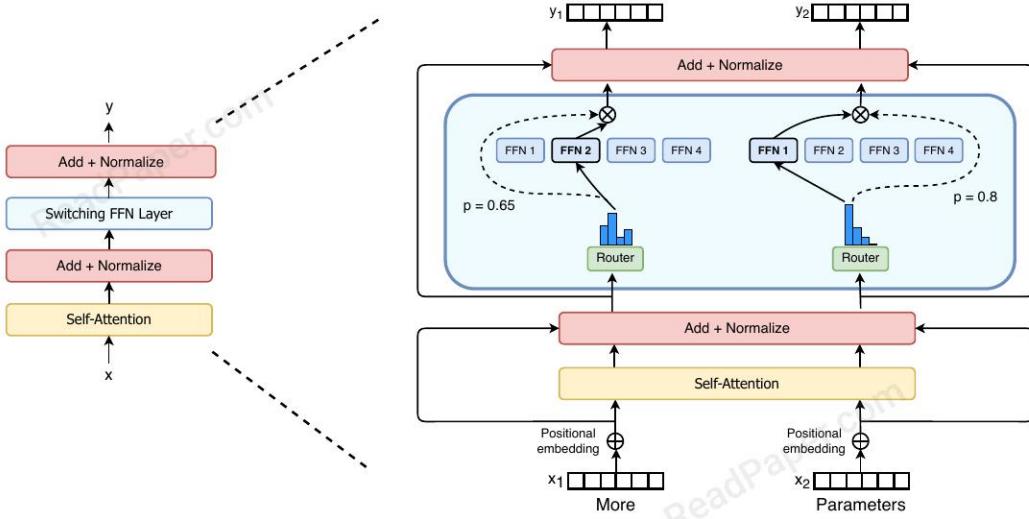


图 2: Switch Transformer 编码器块的图示。我们更换密集饲料具有稀疏开关的 Transformer 中存在前向网络 (FFN) 层 FFN 层 (浅蓝色)。该层独立运行于顺序。我们绘制了两个标记 (下面的 $x_1 = \text{More}$ 和 $x_2 = \text{Parameter}$) 通过四个FFN专家路由 (实线)，其中路由器独立地路由每个令牌。开关FFN层返回所选FFN的输出乘以路由器门值 (虚线)。

2.1 简化稀疏路由

专家路由的混合。沙泽尔等人。(2017) 提出了一种自然语言专家混合 (MoE) 层，它将令牌表示 x 作为输入，然后将其路由到从集合 $\{E_i(x)\}_{i=1}^N$ 中选择的最佳确定的前 k 个专家 N 个专家中的 $i=1$ 。路由器变量 W_r 生成 logits $h(x) = W_r \cdot x$ ，该逻辑通过该层可用的 N 个专家上的 softmax 分布进行归一化。专家 i 的门值由下式给出：

$$p_i(x) = \frac{e^{h(x)_i}}{\sum_j^N e^{h(x)_j}}. \quad (1)$$

选择前 k 个门值来路由令牌 x 。如果 T 是选定的 top- k 索引的集合，则该层的输出计算是每个专家通过门值对令牌进行的计算的线性加权组合，

$$y = \sum_{i \in T} p_i(x) E_i(x). \quad (2)$$

交换机路由：重新思考专家组。沙泽尔等人。(2017) 推测，为了使路由函数具有不平凡的梯度，路由到 $k > 1$ 个专家是必要的。作者凭直觉认为，如果没有能力比较至少两位专家，学习路线就行不通。Ramachandran 和 Le (2018) 进一步指出

研究 top-k 决策，发现模型中较低层的较高 k 值对于具有许多路由层的模型很重要。与这些想法相反，我们使用一种简化的策略，只联系一位专家。我们证明这种简化可以保持模型质量，减少路由计算并且性能更好。这个 $k=1$ 的路由策略稍后被称为交换层。请注意，对于 MoE 和交换机路由，方程 2 中的门值 $p(x)_i$ 允许路由器的可微性。

交换机层的好处有三方面：（1）减少了路由器计算，因为我们只将令牌路由到单个专家。（2）每个专家的批量大小（专家容量）至少可以减半，因为每个令牌仅被路由到单个专家。（3）简化了路由实现并降低了通信成本。图 3 显示了具有不同专家容量因子的路由示例。

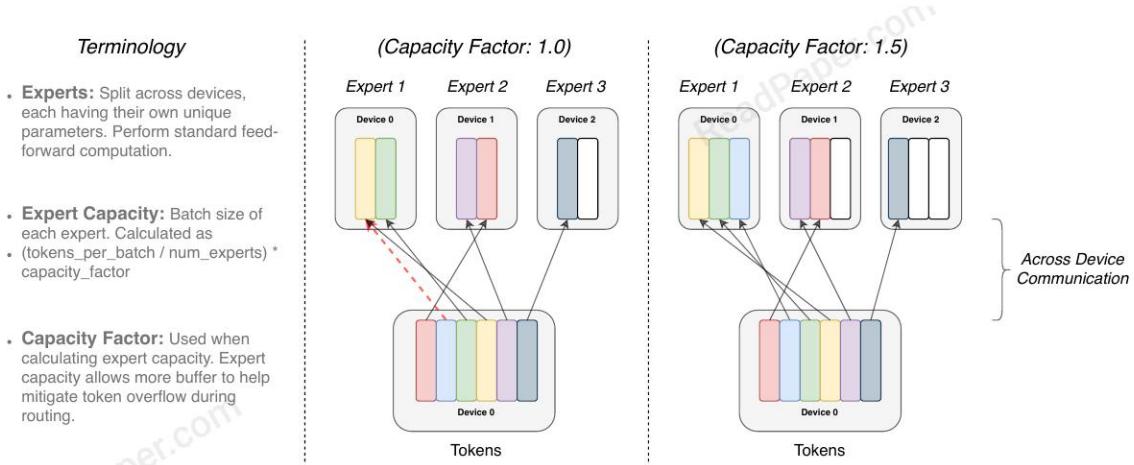


图 3：令牌路由动态图示。每个专家处理固定的批量大小由容量因子调节的代币数量。每个令牌都会路由给专家具有最高的路由器概率，但每个专家都有固定的批量大小 $(\text{总代币} / \text{专家数量}) \times \text{容量系数}$ 。如果代币分配不均匀修补后，某些专家将会溢出（用红虚线表示），导致这些令牌不被该层处理。更大的容量系数减轻解决了这个溢出问题，但也增加了计算和通信成本（由填充的白色/空槽表示）。

2.2 高效稀疏路由

我们使用 Mesh-Tensorflow (MTF) (Shazeer et al., 2018)，它是一个库，具有与 Tensorflow (Abadi et al., 2016) 类似的语义和 API，可促进高效的分布式数据和模型并行架构。它通过将物理核心集抽象为处理器的逻辑网格来实现这一点。然后，张量和计算可以按指定维度进行分片，从而促进跨维度模型的轻松划分。我们在设计模型时考虑到了 TPU，它需要静态声明的大小。下面我们描述我们的分布式 Switch Transformer 实现。

3. 技术说明请参见第 2.2 节。

分布式交换机实施。我们所有的张量形状都是在编译时静态确定的，但由于训练和推理时的路由决策，我们的计算是动态的。因此，如何设置专家能力是一个重要的技术考虑因素。专家容量（每个专家计算的令牌数量）是通过将批次中的令牌数量除以专家数量来设置的，然后进一步按容量因子扩展，

$$\text{专家容量} = \text{每批专家的代币数量} \times \text{容量系数。} \quad (3)$$

大于 1.0 的容量因子会创建额外的缓冲区，以适应专家之间代币未完美平衡的情况。如果太多令牌被路由到专家（稍后称为丢弃令牌），则跳过计算，并且令牌表示通过残差连接直接传递到下一层。然而，增加专家容量并非没有缺点，因为高值会导致计算和内存的浪费。这种权衡在图 3 中得到了解释。根据经验，我们发现确保较低的代币丢弃率对于稀疏专家模型的扩展非常重要。在整个实验过程中，我们没有注意到丢弃的代币数量对专家数量有任何依赖性（通常 < 1%）。使用具有足够高系数的辅助负载平衡损耗（下一节）可确保良好的负载平衡。我们在表 1 中研究了这些设计决策对模型质量和速度的影响。

可微的负载平衡损失。为了鼓励专家之间的平衡负载，我们添加了辅助损失 (Shazeer 等人, 2017、2018; Lepikhin 等人, 2020)。正如 Shazeer 等人所言。(2018)；莱皮欣等人。(2020)，Switch Transformers 简化了 Shazeer 等人的原始设计。(2017)其中有单独的负载平衡和重要性加权损失。对于每个 Switch 层，该辅助损失会在训练期间添加到总模型损失中。给定由 $i = 1$ 到 N 索引的 N 个专家和具有 T 个令牌的批次 B ，辅助损失计算为向量 f 和 P 之间的缩放点积，

$$\text{loss} = \alpha \cdot N \cdot \sum_{i=1}^N f_i \cdot P_i \quad (4)$$

其中 f_i 是分配给专家 i 的代币比例，

$$f_i = \frac{1}{T} \sum_{x \in \mathcal{B}} \mathbb{1}\{\arg\max p(x) = i\} \quad (5)$$

P_i 是分配给专家 i 的路由器概率的分数，2

$$P_i = \frac{1}{T} \sum_{x \in \mathcal{B}} p_i(x). \quad (6)$$

由于我们寻求跨 N 个专家的批量令牌的统一路由，因此我们希望两个向量的值为 $1/N$ 。等式 4 的辅助损耗鼓励均匀路由，因为它在均匀分布下被最小化。目标也可以微分为

2. 一个潜在的混乱来源： $p_i(x)$ 是将令牌 x 路由到专家 i 的概率。 P_i 是专家 i 在批次 B 中所有令牌中的概率分数。

P 向量可微，但 f 向量不可微。最终损失乘以专家计数 N ，以在专家数量变化时保持损失恒定，因为在统一路由下 $\sum_{i=1}^N (f_i \cdot P_i) = \sum_{i=1}^N (1_N \cdot 1_N) = 1_N$ 。最后，超参数 α 是这些辅助损失的乘法系数；在整个工作中，我们使用 $\alpha = 10 \text{ to } 2$ ，它足够大以确保负载平衡，同时又足以不压倒主要交叉熵目标。我们以 10 的幂的形式扫描了从 $10 \text{ to } 1$ 到 $10 \text{ to } 5$ 的 α 超参数范围，并快速找到了 $10 \text{ to } 2$ 平衡负载，而不会干扰训练损失。

2.3 综合起来：开关变压器

我们对 Switch Transformer 的第一次测试从对“Colossal Clean Crawled Corpus”(C4) 的预训练开始，该语料库在(Raïl 等人, 2019 年)中引入。对于我们的预训练目标，我们使用掩码语言建模任务(Taylor, 1953; Fedus 等人, 2018; Devlin 等人, 2018)，其中模型被训练来预测丢失的标记。在我们的预训练环境中，正如 Raïl 等人所确定的那样。(2019)为了达到最佳效果，我们丢弃 15% 的标记，然后用单个哨兵标记替换屏蔽序列。为了比较我们的模型，我们记录了负对数困惑度。⁴ 在本文的所有表格中， $\hat{\alpha}$ 表示该指标的值越高越好，反之亦然。表 9 对本工作中研究的所有模型进行了比较。

表 1 展示了 Switch Transformer 和 MoE Transformer 的正面比较。我们的 Switch Transformer 模型与“T5-Base”进行 FLOP 匹配(Raïl 等人, 2019)(每个模型的计算量相同)应用令牌)。MoE Transformer 使用 top-2 路由，有两个专家，每个专家对每个令牌应用一个单独的 FFN，因此它的 FLOPS 更大。所有模型都在相同的硬件上进行相同数量的训练。请注意，在上述实验设置中，MoE 模型从容量因子 2.0 到 1.25 实际上减慢了速度(840 到 790)，这是出乎意料的。⁵ 我们重点介绍表 1 中的三个关键发现：(1) Switch Transformer 在速度质量基础上优于精心调整的密集模型和 MoE Transformer。对于固定的计算量和挂钟时间，开关变压器可以获得最佳结果。(2) Switch Transformer 的计算量比 MoE 对应物更小。如果我们增加其大小以匹配 MoE Transformer 的训练速度，我们会发现它在每步的基础上也优于所有 MoE 和 Dense 模型。(3) 开关变压器在较低容量系数(1.0、1.25)下表现更好。较小的专家容量表示大型模型体系中的场景，其中模型内存非常稀缺，并且容量因子将希望尽可能大。

2.4 改进的训练和微调技术

稀疏专家模型可能会给普通 Transformer 带来训练困难。由于每一层的硬交换(路由)决策，可能导致不稳定。此外，bfloat16(Wang 和 Kanwar, 2019)等低精度格式可能会加剧问题

4. 我们使用 $\log_{\text{base-e}}$ 作为该指标，因此单位为 nat。

5. 请注意，速度测量既是算法又是实现细节的函数。Switch Transformer 相对于 MoE (算法) 减少了必要的计算，但最终速度差异受到低级优化 (实现) 的影响。

Model	Capacity Factor	Quality after 100k steps (\uparrow) (Neg. Log Perp.)	Time to Quality Threshold (\downarrow) (hours)	Speed (\uparrow) (examples/sec)
T5-Base	—	-1.731	Not achieved [†]	1600
T5-Large	—	-1.550	131.1	470
MoE-Base	2.0	-1.547	68.7	840
Switch-Base	2.0	-1.554	72.8	860
MoE-Base	1.25	-1.559	80.7	790
Switch-Base	1.25	-1.553	65.0	910
MoE-Base	1.0	-1.572	80.1	860
Switch-Base	1.0	-1.561	62.8	1000
Switch-Base+	1.0	-1.534	67.6	780

表 1: 基准交换机与 MoE。每一步的头对头比较测量
 开关变压器相对于 MoE 变压器的单位时间效益
 T5 密集基线。我们通过负对数困惑度和
 达到任意选择的 Neg 质量阈值的时间。对数持久性 = -1.50。全部
 MoE 和 Switch Transformer 模型由 128 名专家组成，每个专家都有
 前馈层。对于 Switch-Base+，我们增加模型大小直到匹配
 通过将模型隐藏大小从 768 增加到 896 来提高 MoE 模型的速度
 头数从 14 到 16。所有模型都使用相同的方法进行训练
 计算量（32 个核心）和相同硬件（TPUv3）。更远
 请注意，我们所有的模型都需要超过 10 万步的预训练才能实现我们的目标
 水平阈值为 -1.50。† T5-Base 没有实现这种负对数困惑
 模型训练的 100k 步。

在我们的路由器的 softmax 计算中。我们在这里描述训练困难以及我们用来克服这些困难以实现稳定和可扩展训练的方法。

大型稀疏模型的选择性精度。模型不稳定性阻碍了使用高效 bfloat16 精度进行训练的能力，因此，Lepikhin 等人。（2020）在其 MoE Transformer 中以 float32 精度进行训练。然而，我们表明，通过在模型的局部部分内选择性地转换为 float32 精度，可以实现稳定性，而不会产生昂贵的 float32 张量通信成本。该技术符合现代混合精度训练策略，其中模型的某些部分和梯度更新是在更高精度中完成的 Micikevicius 等人。

（2017）。表 2 显示，我们的方法允许与 bfloat16 训练几乎相同的速度，同时赋予 float32 训练稳定性。为了实现这一点，我们将路由器输入转换为 float32 精度。路由器函数将令牌作为输入，并生成用于专家计算的选择和重组的调度和组合张量（详细信息请参阅附录中的代码块 15）。重要的是，“float32 精度仅在路由器函数体内”用于该设备本地的计算。因为生成的调度和组合张量在函数结束时被重新转换为 bfloat16 精度，所以没有昂贵的 float32 张量。

Model (precision)	Quality (Neg. Log Perp.) (\uparrow)	Speed (Examples/sec) (\uparrow)
Switch-Base (float32)	-1.718	1160
Switch-Base (bfloat16)	-3.780 [<i>diverged</i>]	1390
Switch-Base (Selective precision)	-1.716	1390

表 2: 选择精度。我们将本地路由操作转换为 oat32，同时保留
在其他地方增加 16 精度以稳定我们的模型，同时实现几乎相同的效果
速度（不稳定）bfloat16 精度训练。我们测量 32 的质量
专家模型在早期训练其速度性能时经过固定步数。为了
oat32 中的 Switch-Base 和 Selective prevision 我们注意到类似的学习
动力学。

通过点对点的通信操作进行广播，但我们仍然受益于 oat32 增强的稳定性。

较小的参数初始化以提高稳定性。适当的初始化对于深度学习的成功训练至关重要，
我们特别观察到这对于 Switch Transformer 来说尤其如此。我们通过从均值 $\mu = 0$ 和标
准差 $\sigma = s/n$ 的截断正态分布中提取元素来初始化权重矩阵，其中 s 是尺度超参数， n
是权重张量中输入单元的数量（例如 fan-in）。作为解决不稳定问题的额外补救措
施，我们建议将默认的 Transformer 初始化比例 $s = 1.0$ 减少 10 倍。这既提高了质
量，又降低了实验中训练不稳定的可能性。表 3 衡量了训练早期模型质量的提高和方差的
减少。我们发现

Model (Initialization scale)	Average Quality (Neg. Log Perp.)	Std. Dev. of Quality (Neg. Log Perp.)
Switch-Base (0.1x-init)	-2.72	0.01
Switch-Base (1.0x-init)	-3.60	0.68

表 3: 减小初始化规模可提高稳定性。减少初始化规模
带来更好的模型质量和更稳定的 Switch Transformer 训练。
在这里，我们记录模型质量的平均值和标准偏差，通过以下方法测量
32 个专家模型经过 3.5k 个步骤（3 个随机种子）后的负对数困惑度
每个）。

平均模型质量，由 Neg 测量。Log Perp. 得到显着改善，并且运行之间的方差大大减少。
此外，这种相同的初始化方案对于跨越几个数量级的模型广泛有效。我们使用相同的方法
来稳定地训练模型，从 223M 参数基线到超过 1 万亿参数的庞大模型。

6. 对平均值大于两个标准差的值进行重新采样。

规范大型稀疏模型。我们的论文考虑了常见的 NLP 方法，即在大型语料库上进行预训练，然后对所有下游任务（例如摘要或问答）进行微调。自然出现的一个问题是过度拟合，因为许多微调任务的示例很少。在标准 Transformer 的微调过程中，Raïl 等人。(2019) 在每一层使用 dropout (Srivastava et al., 2014) 来防止过度拟合。我们的 Switch Transformer 比 FLOP 匹配的密集基线具有更多的参数，这可能会导致这些更严重的下游任务出现更严重的过度。

Model (dropout)	GLUE	CNNDM	SQuAD	SuperGLUE
T5-Base (d=0.1)	82.9	19.6	83.5	72.4
Switch-Base (d=0.1)	84.7	19.1	83.7	73.0
Switch-Base (d=0.2)	84.4	19.2	83.9	73.2
Switch-Base (d=0.3)	83.9	19.6	83.4	70.7
Switch-Base (d=0.1, ed=0.4)	85.2	19.6	83.7	73.0

表 4：微调正则化结果。微调时对辍学率进行扫描

Switch Transformer 模型在 C4 数据集的 34B 代币上进行预训练（更高数字更好）。我们观察到，使用较低的标准辍学率所有非专家层，专家前馈的辍学率要大得多层层叠加，发挥最佳性能。

因此，我们提出了一种简单的方法来缓解调优过程中的这个问题：增加专家内部的dropout，我们将其称为专家dropout。在微调期间，我们仅在每个专家层的临时前馈计算中将丢失率显着增加。表 4 列出了我们的专家 dropout 协议的结果。我们观察到，简单地增加所有层的 dropout 会导致性能更差。然而，在非专家层设置较严重的丢失率(0.1) 和在专家层设置较大的丢失率(0.4) 会导致四个较严重的下游任务的性能提高。

3. 缩放属性

我们对预训练期间 Switch Transformer 架构的缩放特性进行了研究。每卡普兰等人。(2020)，我们考虑了一种模型不受计算预算或数据量瓶颈的机制。为了避免数据瓶颈，我们使用包含超过 180B 个目标标记的大型 C4 语料库 (Raïl 等人, 2019) 并进行训练，直到观察到收益递减。

专家的数量是扩展我们模型的最有效的维度。增加专家可以使计算成本保持大致固定，因为模型只为每个令牌选择一名专家，而不管可供选择的专家数量如何。路由器必须计算更多专家的概率分布，但是，这是成本 $O(d \text{ 模型} \times \text{专家数})$ 的轻量级计算，其中 d 模型是的嵌入维度

令牌在各层之间传递。在本节中，我们在固定计算预算的情况下考虑步进基础和时间基础上的缩放属性。

3.1 逐步缩放结果

图 4 展示了在针对固定步骤数训练所有模型时，随着专家数量的增加，具有一致的扩展优势。我们观察到一个明显的趋势：当保持每个令牌的 FLOPS 固定时，拥有更多参数（专家）可以加快训练速度。左图展示了稀疏模型参数和测试损失之间一致的缩放特性（每个令牌的 FLOPS 固定）。这揭示了沿着稀疏模型参数的这个附加轴进行缩放的优势。我们的右图测量了密集模型变体和四个 FLOP 匹配的稀疏变体的样本效率。我们发现，增加专家数量会带来样本效率更高的模型。我们的 Switch-Base 64 专家模型在步骤 60k 和步骤 450k 上实现了与 T5-Base 模型相同的性能，这在步骤时间方面加速了 7.5 倍。此外，与 Kaplan 等人的研究结果一致。（2020），我们发现较大的模型样本效率也更高，对于固定数量的观察到的标记可以更快地学习。

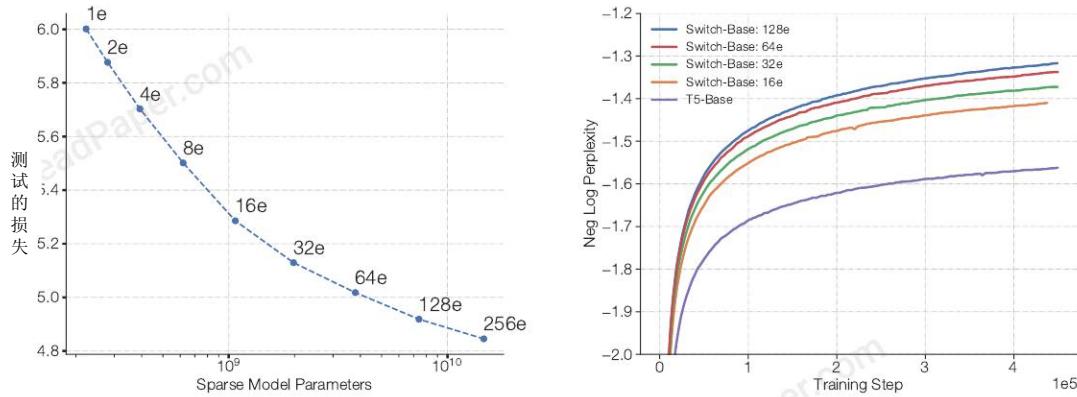


图 4：开关变压器的缩放属性。左图：我们衡量质量随着参数通过缩放而增加，通过困惑度来衡量的改进专家数量。左上角的点对应于 T5-Base 模型 223M 参数。从左上角到右下角，我们将数量加倍专家从 2、4、8 等直到 256 专家模型的右下点具有 14.7B 参数。尽管所有模型都使用相同的计算预算，我们观察到专家数量不断增加。右图：每一步的负对数困惑席卷了专家的数量。密集的基线用紫色线显示，我们注意到样本效率的提高我们的开关底座型号。

开关变压器

3.2 按时间缩放结果

图 4 表明，随着专家数量的增加，性能会逐步提高。虽然我们的模型每个代币的 FLOPS 量与基准大致相同，但我们的 Switch Transformer 会产生跨设备的额外通信成本以及路由机制的额外计算。因此，逐步观察到的样本效率的提高并不一定会转化为通过挂钟测量的更好的模型质量。这就提出了一个问题：

对于固定的训练时间和计算预算，应该训练密集模型还是稀疏模型？

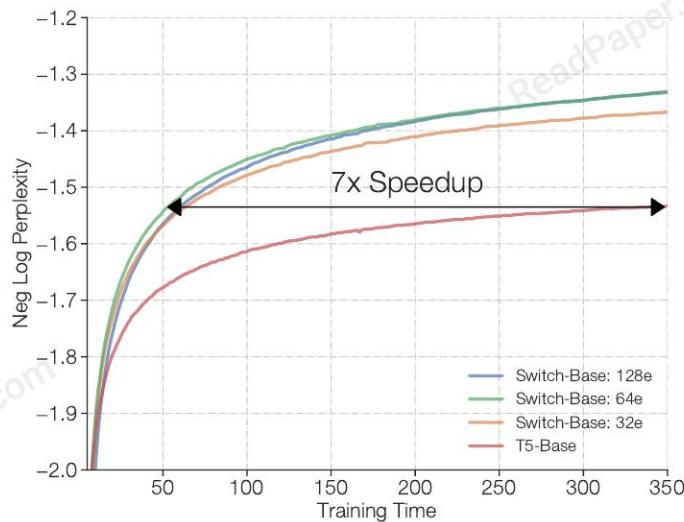


图 5：开关变压器的速度优势。所有模型均在 32 个 TPUv3 核心上进行训练
每个示例的 FLOP 次数相同。对于固定量的计算和训练
时间，开关变压器显着优于密集变压器基础
线。我们的 64 名专家 Switch-Base 模型只需七分之一的时间即可达到相同的质量
T5-Base 的时间并持续改进。

图 5 和图 6 解决了这个问题。图 5 测量了预训练模型质量随时间的变化。对于固定的训练时间和计算预算，Switch Transformers 可以显着提高速度。在此设置中，我们的 Switch-Base 64 专家模型的训练时间是 T5-Base 获得类似困惑所需时间的七分之一。

3.3 缩放与更大的密集模型

上述分析表明，计算匹配的密集模型的速度超过了 Switch 对应模型。图 6 考虑了一个不同的场景：如果我们将资源分配给更大的密集模型会怎样？我们现在这样做，根据下一个强基线 T5-Large 来衡量 Switch-Base。但是，尽管 T5-Large 每个代币的 FLOP 数增加了 3.5 倍，

Switch-Base仍然是更多的样本，并产生了2.5倍的加速。此外，更多的收益可以简单地通过设计一个新的，更大的稀疏版本，开关-大，它是flop匹配的t5-大。我们将在下面的小节中演示高级伸缩和天真调优。

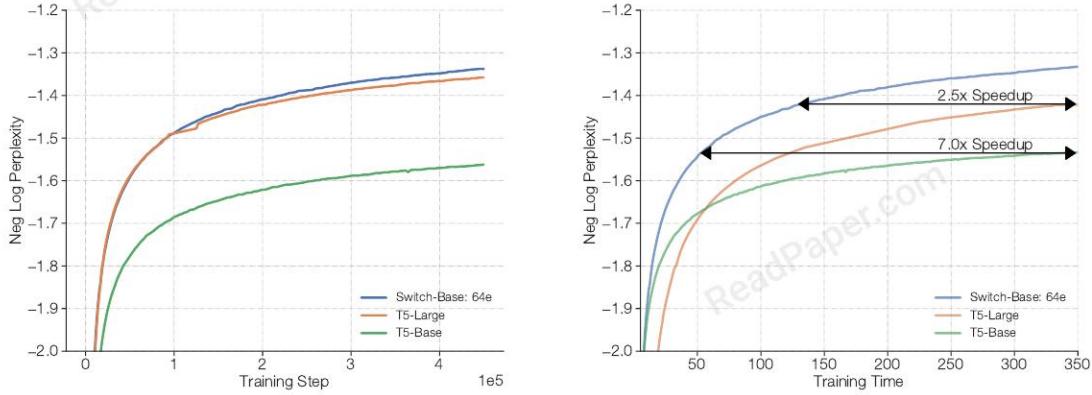


图 6：使用 Switch 层或标准密集模型缩放 Transformer 模型
缩放。左图：Switch-Base 比 T5-Base 的样本效率更高，
T5-Large 变体，每个代币的 FLOPS 提高了 3.5 倍。右图：如
之前，根据挂钟的基础，我们发现 Switch-Base 仍然更快，并且产生
比 T5-Large 加速 2.5 倍。

4. 下游结果

第 3 节展示了预训练时的卓越扩展特性，但我们现在验证这些收益可以转化为下游任务中语言学习能力的提高。我们首先对一组不同的 NLP 任务进行微调。接下来，我们研究通过提炼成所有“且易于部署”的密集基线，将稀疏模型的内存占用量减少 90% 以上。最后，我们在本节的结论中衡量了多任务、多语言环境中的改进，其中我们表明 Switch Transformers 是强大的多任务学习器，在所有 101 种语言中比基于 T5 的多语言模型有所改进。

4.1 Fine-Tuning

用于微调的基线和开关模型。我们的基线是高度调整的 223M 参数 T5-Base 模型和 739M 参数 T5-Large 模型 (Raïl 等人, 2019)。对于这两个版本，我们设计了一个 FLOP 匹配的开关变压器，具有更多参数，表 9 中总结了这些参数。⁷ 我们的基线与 Raïl 等人的基线略有不同。(2019) 因为我们在改进的 C4 语料库上进行预训练，该语料库消除了示例内的文本重复，从而提高了预训练任务的有效性。

7. FLOPS 是按照 Kaplan 等人的方法计算前向传播的。(2020)。

(2021)。在我们的协议中，我们每批使用 2~20 (1,048,576) 个令牌进行 550k 个步骤的预训练，总计 576B 个令牌。然后，我们对除 Switch 层之外的所有层使用 0.1 的 dropout 率对一组不同的任务进行微调，Switch 层使用 0.4 的 dropout 率（参见表 4）。我们使用 1M 的批量大小进行 16k 个步骤的微调，对于每个任务，我们每 200 个步骤评估一次模型质量，并报告在验证集上计算的峰值性能。

微调任务和数据集。我们选择探索语言能力的任务，包括回答问题、总结和关于世界的知识。语言基准 GLUE (Wang 等人, 2018) 和 SuperGLUE (Wang 等人, 2019) 被作为复合混合物处理，所有任务按每个任务中存在的标记数量的比例混合。这些基准包括需要情感分析 (SST-2)、词义消歧 (WIC)、句子相似度 (MRPC、STS-B、QQP)、自然语言推理 (MNLI、QNLI、RTE、CB)、问答 (Multirc) 的任务，RECORD, BoolQ)，共指消解 (WNLI, WSC) 和句子完成 (COPA) 和句子可接受性 (CoLA)。CNNDM (Hermann et al., 2015) 和 BBC XSum (Narayan et al., 2018) 数据集用于衡量总结文章的能力。使用 SQuAD 数据集 (Rajpurkar 等人, 2016) 和 ARC 推理挑战赛 (Clark 等人, 2018) 来探讨问题回答。正如罗伯茨等人所言。(2020)，我们通过对三个闭卷问答数据集进行微调来评估我们模型的知识：自然问题 (Kwiatkowski 等人, 2019)、网络问题 (Berant 等人, 2013) 和 Trivia质量保证 (Joshi 等人, 2017)。闭卷是指在没有补充参考资料或背景材料的情况下提出的问题。为了衡量模型的常识推理，我们在 Winogrande Schema Challenge 上对其进行评估 (Sakaguchi 等人, 2020)。最后，我们在对抗性 NLI 基准测试中测试了我们模型的自然语言推理能力 (Nie 等人, 2019)。

微调指标。整篇论文使用了以下评估指标：我们报告了 GLUE 和 SuperGLUE 所有子任务的平均分数。Rouge-2 度量同时用于 CNNDM 和 XSum。在 SQuAD 和闭卷任务（网络问题、自然问题和琐事问题）中，我们报告了与目标完全匹配的答案的百分比（请参阅 Roberts 等人 (2020) 了解此衡量标准的更多详细信息和缺陷）。最后，在 ARC Easy、ARC Challenge、ANLI 和 Winogrande 中，我们报告了生成的响应的准确性。

微调结果。我们观察到许多自然语言任务的下游显着改进。显着的改进来自 SuperGLUE，我们发现 FLOP 匹配的 Switch 变体分别比 T5-Base 和 T5-Large 基线提高了 4.4 个百分点和 2 个百分点，并且 Winogrande、闭卷 Trivia QA 和 XSum 也有很大改进。⁸ 在我们的微调研究中，我们唯一没有观察到增益的任务是在 AI2 Reasoning Challenge (ARC) 数据集上，其中 T5-Base 在挑战数据集上的表现优于 Switch-Base，T5-Large 的表现优于 Switch -大量的简单数据集。总的来说，我们观察到推理和知识密集型任务都取得了显着的进步。这验证了我们的架构，不仅是预训练良好的架构，而且可以通过微调将质量改进转化为下游任务。

8. 我们的 T5 和 Switch 模型在修订后的 C4 数据集上使用每批 2~20 个令牌进行了 55 万步的预训练，以进行公平比较。

Model	GLUE	SQuAD	SuperGLUE	Winogrande (XL)
T5-Base	84.3	85.5	75.1	66.6
Switch-Base	86.7	87.2	79.5	73.3
T5-Large	87.8	88.1	82.7	79.1
Switch-Large	88.5	88.6	84.7	83.0

Model	XSum	ANLI (R3)	ARC Easy	ARC Chal.
T5-Base	18.7	51.8	56.7	35.5
Switch-Base	20.3	54.0	61.3	32.8
T5-Large	20.9	56.6	68.8	35.5
Switch-Large	22.3	58.6	66.0	35.5

Model	CB Web QA	CB Natural QA	CB Trivia QA
T5-Base	26.6	25.8	24.5
Switch-Base	27.4	26.8	30.7
T5-Large	27.7	27.6	29.5
Switch-Large	31.3	29.5	36.9

表 5: 微调结果。T5 基线和 Switch 模型的微调结果

一组多样化的自然语言测试（验证集；数字越高越好）。

我们将 FLOP 匹配的 Switch 模型与 T5-Base 和 T5-Large base-进行比较
线。对于大多数考虑的任务，我们发现 Switch 有了显着的改进
变种。我们观察到模型大小以及推理和推理方面的收益
知识密集型语言任务。

4.2 蒸馏

部署具有数十亿或数万亿参数的大规模神经网络非常不方便。为了缓解这个问题，我们研究将大型稀疏模型蒸馏 (Hinton et al., 2015) 为所有密集模型。未来的工作还可以研究将大型模型提炼为更稀疏的模型。

蒸馏技术。在表 6 中，我们研究了多种蒸馏技术。这些技术是由 Sanh 等人构建的。(2019)，研究 BERT 模型的蒸馏方法。我们发现，使用非专家权重初始化密集模型会产生适度的改进。这是可能的，因为所有模型都是 FLOP 匹配的，因此非专家层将具有相同的尺寸。由于专家层通常仅添加到 Transformer 中的每个或每个其他 FFN 层，因此这允许使用经过训练的参数来初始化许多权重。此外，我们观察到使用 0.25 的教师概率和 0.75 的地面真值标签的混合进行蒸馏改进。通过结合这两种技术，我们仅用 1/20 的参数就保留了大型稀疏模型 30% 的质量增益。质量增益指的是

Switch-Base (教师) 和 T5-Base (学生) 之间的质量差异。因此, 100% 的质量增益意味着学生的表现等于教师的表现。

Technique	Parameters	Quality (\uparrow)
T5-Base	223M	-1.636
Switch-Base	3,800M	-1.444
Distillation	223M	(3%) -1.631
+ Init. non-expert weights from teacher	223M	(20%) -1.598
+ 0.75 mix of hard and soft loss	223M	(29%) -1.580
Initialization Baseline (no distillation)		
Init. non-expert weights from teacher	223M	-1.639

表 6: 提取用于语言建模的开关变压器。初始化 T5-Base 来自 Switch-Base 的非专家权重并使用教师混合的损失和真实标签获得最佳性能。我们可以蒸馏出 30% 参数增加 100 倍的大型稀疏模型的性能改进进入全密集模型。对于最终基线, 我们发现 T5-Base 没有任何改进使用专家重量进行初始化, 但无需蒸馏即可正常训练。

可实现的压缩率。使用表 6 中描述的最佳蒸馏技术, 我们将各种稀疏模型蒸馏为密集模型。我们提炼了 Switch-Base 版本, 涵盖了越来越多的专家, 这对应于 1.1B 到 14.7B 参数之间的变化。通过蒸馏, 我们可以在压缩 82% 的同时保留 1.1B 参数模型 37% 的质量增益。在极端情况下, 当我们将模型压缩 99% 时, 我们仍然能够保持教师模型质量提高 28%。

提炼出一个经过微调的模型。我们通过将微调稀疏模型提炼为稠密模型的研究来得出结论。表 8 显示了将 7.4B 参数 Switch-Base 模型 (在 SuperGLUE 任务上进行微调) 提炼到 223M T5-Base 中的结果。与我们的预训练结果类似, 我们发现, 当提炼成 FLOP 匹配的密集变体时, 我们能够保留稀疏模型 30% 的增益。一种潜在的未来途径 (此处未考虑) 可能会检查用于微调任务的特定专家并提取它们以实现更好的模型压缩。

4.3 多语言学习

在我们最后一组下游实验中, 我们在对 101 种不同语言进行预训练时衡量模型质量和速度的权衡。我们对 mT5 (Xue 等人, 2020) 的最新工作进行了构建和基准测试, mT5 是 T5 的多语言扩展。我们对 Common Crawl 数据集 (mC4) 的多语言变体进行预训练, 该变体涵盖 mT5 中引入的 101 种语言, 但由于某些语言内的脚本变体, 该混合物包含 107 个任务。

在图 7 中, 我们绘制了 FLOP 匹配 Switch 模型 (mSwitch-Base) 到 T5 基本变体 (mT5-Base) 的所有语言的负对数困惑度的质量改进。后

	Dense	Sparse				
Parameters	223M	1.1B	2.0B	3.8B	7.4B	14.7B
Pre-trained Neg. Log Perp. (\uparrow)	-1.636	-1.505	-1.474	-1.444	-1.432	-1.427
Distilled Neg. Log Perp. (\uparrow)	—	-1.587	-1.585	-1.579	-1.582	-1.578
Percent of Teacher Performance	—	37%	32%	30 %	27 %	28 %
Compression Percent	—	82 %	90 %	95 %	97 %	99 %

表 7: 蒸馏压缩率。我们在蒸馏大稀疏时测量质量

模型进入密集基线。我们的基线 T5-Base 的负值为 -1.636。日志 Perp。质量。然后，在右列中，我们提取越来越大的稀疏模型进入同一个架构。通过权重初始化和硬损失和软损失的混合，我们可以将稀疏的教师人数减少 95%+ 同时保留 30% 的质量增益。然而，为了显着更好和规模更大的预培训教师，我们预计需要更大的学生模型达到这些压缩率。

Model	Parameters	FLOPS	SuperGLUE (\uparrow)
T5-Base	223M	124B	74.6
Switch-Base	7410M	124B	81.3
Distilled T5-Base	223M	124B	(30%) 76.6

表 8: 提炼出经过微调的 SuperGLUE 模型。我们提炼出一个 Switch-Base 模型 fine- 将 SuperGLUE 任务调整为 T5-Base 模型。我们观察到，在所有数据集 我们的大型稀疏模型可以成为蒸馏的有效老师。我们

◆ 我们再次在 97% 压缩的情况下实现了教师表现的 30% 模型。

对两个版本进行 1M 步骤的预训练，我们发现在考虑的所有 101 种语言中，Switch Transformer 增加了最终负对数困惑度超过基线。在图 8 中，我们提出了一个不同的视图，现在用直方图绘制了使用 Switch Transformer 相对于 mT5-Base 的每步加速情况。9 我们发现 mT5-Base 的平均加速比为 5 倍，并且 91% 的语言至少实现了 4 倍的加速。这证明 Switch Transformers 是有效的多任务和多语言学习者。

5. 使用数据、模型和专家并行设计模型

任意增加专家数量会导致收益递减（图 4）。在这里，我们描述互补的扩展策略。缩放 Transformer 的常见方法是串联增加维度，例如 d_{model} 或 d_{ff} 。这增加了两个参数

9. 基于步骤的加速计算为基线步骤数除以我们的模型达到相同质量所需的步骤数的比率。

开关变压器

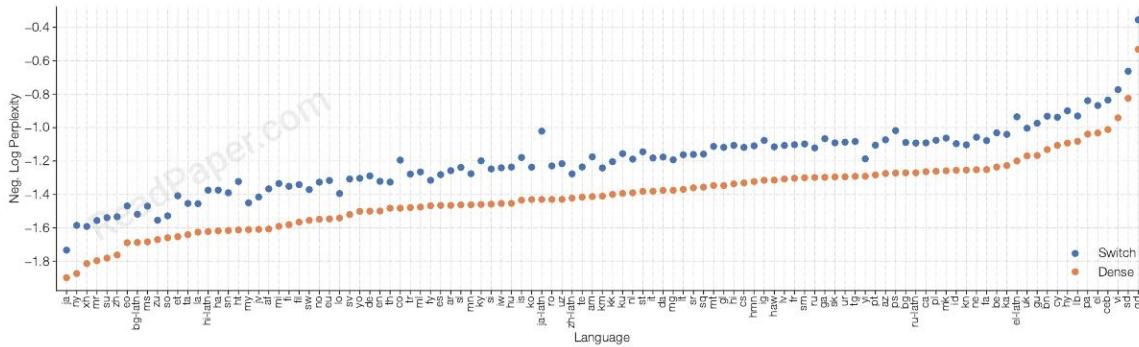


图 7: 101 种语言的多语言预训练。Switch T5底座的改进

在 101 种语言上进行多任务训练时，模型超过了密集基线。我们观察到 Switch Transformers 在多任务训练设置和产量方面表现良好对所有 101 种语言的改进。

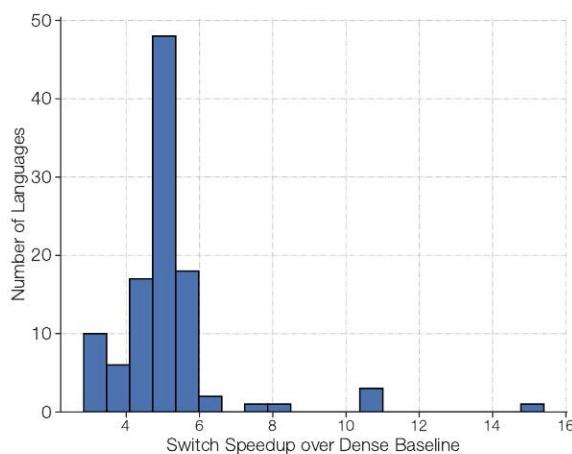


图 8: 101 种语言的多语言预训练。我们为每种语言绘制直方图，Switch Transformer 相对于 FLOP 匹配的 T5 密集基线的步进加速达到同样的品质。在所有 101 种语言中，我们实现了平均步速 – 比 mT5-Base 提高了 5 倍，对于 91% 的语言，我们记录了 4 倍或更高，加速达到 mT5-Base 的最终困惑度。

执行的计算最终受每个加速器的内存限制。一旦超过加速器内存的大小，就可以采用单程序多数据 (SPMD) 模型并行。本节研究数据、模型和专家并行相结合的权衡。

回顾前馈网络 (FFN) 层。我们使用 FFN 层作为数据、模型和专家并行如何在 Mesh TensorFlow 中工作的示例 (Shazeer 等人, 2018)，并在此对其进行简要回顾。我们假设批次中有 B 个令牌，每个维度

d 模型。FFN 的输入 (x) 和输出 (y) 的大小均为 $[B, d_{model}]$ ，中间值 (h) 的大小为 $[B, d_{ff}]$ ，其中 d_{ff} 通常比 d_{model} 大几倍。在 FFN 中，中间层是 $h = xW_{in}$ ，然后该层的输出是 $y = \text{ReLU}(h)W_{out}$ 。因此， W_{in} 和 W_{out} 独立地应用于每个标记，并且具有大小 $[d_{model}, d_{ff}]$ 和 $[d_{ff}, d_{model}]$ 。

我们描述了分区的两个方面：数据的权重和批次如何在核心上划分，如图 9 所示。我们将所有可用核心表示为 N ，然后网格张量流可以将其重新映射到处理器的逻辑多维网格中。这里我们创建一个二维逻辑网格，一维表示数据并行分片的路数 (n)，另一维表示模型并行分片的路数 (m)。总核心数必须等于数据和模型并行的分片方式，例如 $N = n \times m$ 。为了跨核心对层进行分片，包含该批次 B 令牌的张量将被分片到 n 个数据并行核心，因此每个核心都包含 B/n 个令牌。然后，具有 d_{ff} 的张量和变量被分片到 m 个模型并行核心上。对于具有专家层的变体，我们考虑 E 个专家，每个专家最多可以处理 C 个令牌。

术语说明

B 批次中的令牌数量。 N 总核心数。 n 数据并行分片的方式数量。 m 模型并行分片的方式数量。 E 交換机层专家数量。 C 专家容量，每个专家的批量大小。

。

5.1 数据并行

当训练数据并行模型时，这是分布式训练的标准，那么所有核心都被分配到数据并行维度或 $n = N, m = 1$ 。这样做的优点是，在整个前向和后向传递之前不需要通信完成后，需要在所有核心上聚合梯度。这对应于图 9 的最左列。

5.2 模型并行

我们现在考虑一种场景，其中所有核心都专门分配给模型并行维度，因此 $n = 1, m = N$ 。现在，所有核心都必须保留完整的 B 令牌，并且每个核心将包含唯一的权重切片。对于每次前向和后向传递，现在都会产生通信成本。每个核心发送 $[B, d_{model}]$ 的张量来计算第二个矩阵乘法 $\text{ReLU}(h)W_{out}$ ，因为 d_{ff} 维度已分区并且必须求和。作为一般规则，每当必须对跨核划分的维度求和时，就会为前向和后向传递添加全归约操作。这与纯数据并行形成对比，纯数据并行仅在整个前向和后向传递结束时发生全归约。

开关变压器

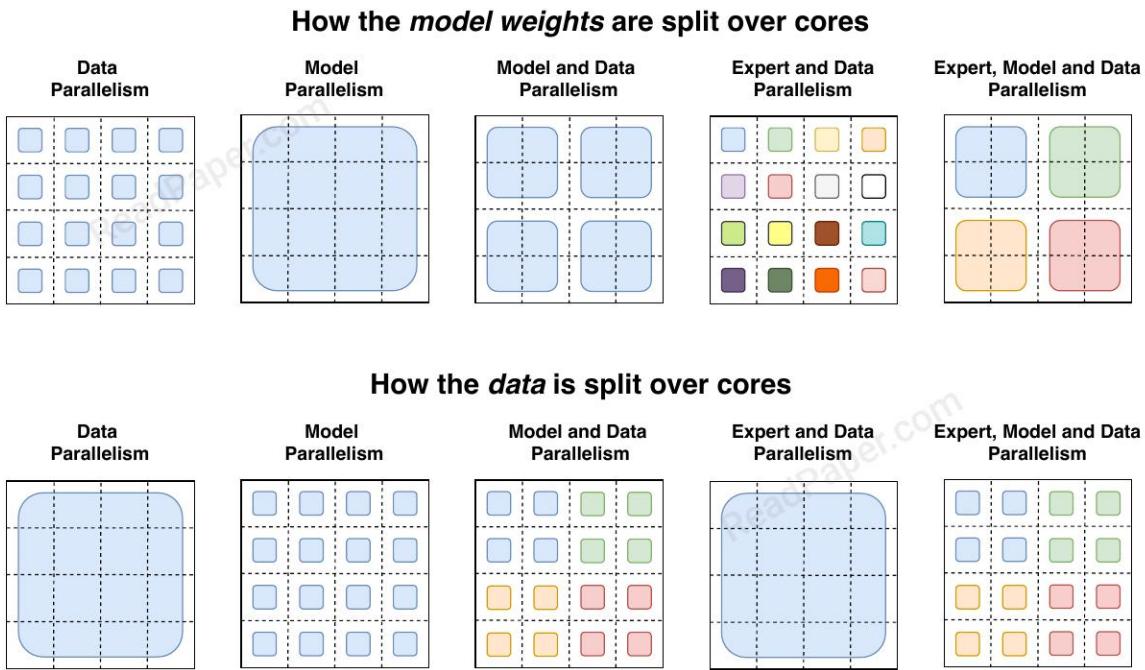


图 9: 数据和权重划分策略。每个 4×4 的虚线网格代表16核心和阴影方块是该核心上包含的数据（无论是模型权重或一批令牌）。我们说明了模型的权重和数据张量针对每个策略进行分割。第一行：模型的说明权重分布在各个核心上。该行中不同大小的形状代表前馈网络 (FFN) 层中较大的权重矩阵（例如较大的 dff 尺寸）。阴影方块的每种颜色都标识一个唯一的权重矩阵。这每个核心的参数数量是固定的，但将应用更大的权重矩阵对每个标记进行更多计算。第二行：数据的说明批次被分割到多个核心上。每个核心持有相同数量的令牌在所有策略中保持固定的内存使用量。分区策略具有不同的属性，允许每个核心具有相同的令牌或核心之间有不同的标记，这就是不同颜色所代表的含义。

5.3 模型和数据并行

对于大规模模型，混合模型和数据并行是很常见的，这在最大的 T5 模型 (Raffel et al., 2019; Xue et al., 2020) 和 GPT-3 (Brown et al., 2020) 中完成。总共有 $N = n \times m$ 个核心，现在每个核心将负责 B/n 代币以及权重和中间激活的 d/mff 。在前向和后向传递中，每个核在全归约操作中传递大小为 $[B/n, d]$ 模型的张量。

5.4 专家和数据并行

接下来我们描述专家和数据并行的分区策略。Switch Transformers 将把它们的所有核心分配给数据划分维度 n ，这也对应于模型中专家的数量。对于每个核心的每个令牌，路由器都会在本地计算分配给专家的分配。输出是大小为 $[n, B/n, E, C]$ 的二进制矩阵，该矩阵在第一维上进行划分并确定专家分配。然后使用该二进制矩阵通过与 $[n, B/n, d_{model}]$ 的输入张量的矩阵乘法进行收集。

$$\text{einsum}([n, B/n, d_{model}], [n, B/n, E, C], \text{dimension} = [B/n]) \quad (7)$$

产生形状为 $[n, E, C, d_{model}]$ 的最终张量，该张量在第一维度上进行分片。因为每个核心都有自己的专家，所以我们进行 $[E, C, d_{model}]$ 大小的全面通信，现在对 E 维度而不是 n 维度进行分片。在前向传递中，为了类似地从位于不同核心上的每个专家接收令牌，需要额外的大小为 $E \times C \times d_{model}$ 的 16 个张量的通信成本。专家分区代码的详细分析请参见附录 F。

5.5 专家、模型和数据并行

在设计最佳模型时，我们寻求平衡每个代币的 FLOPS 和参数计数。当我们扩大专家数量时，我们会增加参数数量，但不会改变每个代币的 FLOP 数。为了增加 FLOP，我们还必须增加 d_{ff} 维度（这也会增加参数，但速度较慢）。这提出了一个权衡：当我们增加 d_{ff} 时，我们将耗尽每个核心的内存，从而需要增加 m 。但由于我们有固定数量的核心 N ，并且 $N = n \times m$ ，因此我们必须减少 n ，这会强制使用更小的批量大小（以便保持每个核心的令牌不变）。

当将模型和专家并行相结合时，我们将拥有将令牌路由到正确专家的所有通信成本以及来自模型并行的内部全归约通信。当结合所有三种方法时，根据经验确定最佳映射，平衡 FLOPS、通信成本和每个内核的内存变得相当复杂。请参阅第 5.6 节中的进一步分析，了解专家数量如何影响下游绩效。

5.6 迈向万亿参数模型

结合专家、模型和数据并行，我们设计了两个大型 Switch Transformer 模型，一个分别具有 3950 亿和 1.6 万亿个参数。我们研究这些模型在作为语言模型的上游预训练及其下游微调性能方面的表现。表 9 列出了两个不同模型的参数、每个序列的 FLOP 和超参数。描述了 Transformer 的标准超参数，包括 d_{model} 、 d_{ff} 、 d_{kv} 、头数和层数，以及一个不太常见的特征 FFN_GELU，它指的是 FFN 层的变体，其中扩展矩阵被非线性组合的两组权重代替 (Shazeer, 2020)。

Switch-C 模型的设计仅使用专家并行，而不使用模型并行，如前面第 5.4 节所述。因此，控制宽度的超参数，

开关变压器

Model	Parameters	FLOPs/seq	d_{model}	FFN_{GEGLU}	d_{ff}	d_{kv}	Num. Heads
T5-Base	0.2B	124B	768	✓	2048	64	12
T5-Large	0.7B	425B	1024	✓	2816	64	16
T5-XXL	11B	6.3T	4096	✓	10240	64	64
Switch-Base	7B	124B	768	✓	2048	64	12
Switch-Large	26B	425B	1024	✓	2816	64	16
Switch-XXL	395B	6.3T	4096	✓	10240	64	64
Switch-C	1571B	890B	2080		6144	64	32
Model	Expert Freq.	Num. Layers	Num Experts	Neg. Log Perp. @250k	Neg. Log Perp. @ 500k		
T5-Base	–	12	–	-1.599	-1.556		
T5-Large	–	24	–	-1.402	-1.350		
T5-XXL	–	24	–	-1.147	-1.095		
Switch-Base	1/2	12	128	-1.370	-1.306		
Switch-Large	1/2	24	128	-1.248	-1.177		
Switch-XXL	1/2	24	64	-1.086	-1.008		
Switch-C	1	15	2048	-1.096	-1.043		

表 9: Switch 模型设计和预训练性能。我们比较超 T5 模型的参数和预训练性能到我们的 Switch Transformer 以前的变种。最后两列记录了预训练模型的质量，分别经过 250k 和 500k 步后的 C4 数据集。我们观察到开关- C Transformer 变体的速度比固定困惑度快 4 倍（使用相同的计算预算）高于 T5-XXL 模型，随着训练的进行，差距越来越大。

深度、头数等都比 T5-XXL 型号要好很多。相比之下，Switch-XXL 与 T5-XXL 模型是 FLOP 匹配的，这允许更大尺寸的超参数，但代价是模型并行引起的额外通信成本（更多详细信息，请参阅第 5.5 节）。

样品效率与 T5-XXL 的比较。在表 9 的最后两列中，我们分别记录了 25 万步和 50 万步后 C4 语料库上的负对数困惑度。经过 25 万步后，我们发现两种 Switch Transformer 变体都比 T5-XXL 版本的负对数困惑度提高了 0.061 以上。¹⁰ 为了说明 0.061 差距的重要性，我们注意到 T5-XXL 模型必须额外训练 25 万步才能增加 0.052。随着额外的训练，差距继续扩大，Switch-XXL 模型的性能比 T5-XXL 好 0.087，步数为 500k。训练不稳定。然而，正如简介中所述，大型稀疏模型可能不稳定，并且随着规模的增加，我们会遇到一些零星的问题。我们发现，具有 1.6T 参数和 2048 名专家的较大 Switch-C 模型根本没有表现出训练不稳定。相反，Switch XXL 版本的每个序列的 FLOP 增加了近 10 倍，有时会不稳定。因此，虽然这是我们在步骤基础上更好的模型，但我们没有预训练完整的 1M 步骤，这与 T5 的最终报告结果一致（Raïl 等人，2019）。

¹⁰. 所报告的质量差异是一个下限，实际上可能更大。 T5-XXL 在更简单的 C4 数据集上进行了预训练，其中包括示例中重复且易于复制的片段。

推理和调整性能。作为对模型质量的初步评估，我们使用在 503B 令牌（大约是 T5-XXL 模型使用的文本的一半）上进行部分预训练的 Switch-XXL 模型。使用这个检查点，我们进行多任务训练以提高效率，其中所有任务都是共同学习的，而不是单独调整的。我们发现验证集上的 SQuAD 准确度提高到 89.7，而最先进的准确度为 91.3。接下来，SuperGLUE 测试的平均得分为 87.5，而 T5 版本的得分为 89.3，而最先进的得分为 90.0 (Wang 等人, 2019)。在 ANLI (Nie 等人, 2019) 上，Switch XXL 比之前最先进的技术有所改进，获得了 65.7 的准确度，而之前的最佳准确度为 49.4 (Yang 等人, 2020)。我们注意到，虽然 Switch- XXL 拥有最先进的 Neg. 日志 Perp. 在上游预训练任务上，其收益尚未完全转化为 SOTA 下游性能。我们在附录 E 中详细研究了这个问题。

基于知识的性能微调。最后，我们还通过三个闭卷的基于知识的任务对模型的知识进行了早期检查：自然问题、WebQuestions 和 TriviaQA，无需使用显着跨度掩码进行额外的预训练 (Guu 等人, 2020)。在所有三种情况下，我们都观察到相对于先前最先进的 T5-XXL 模型（无 SSM）的改进。自然问题的精确匹配度从之前的 32.8 提高到 34.4，网络问题从 37.2 提高到 41.0，TriviaQA 则从 42.9 提高到 47.5。总而言之，尽管训练数据少于其他模型的一半，但我们已经找到了可比的、有时甚至是最先进的模型质量。目前，Switch Transformer 将大量上游收益更好地转化为基于知识的任务，而不是推理任务（参见附录 E）。从大型专家模型中提取更强的微调性能是一个活跃的研究问题，预训练的困惑表明未来的改进应该是可能的。

六、相关工作

神经网络中规模的重要性得到了广泛认可，并且已经提出了几种方法。最近的工作通过使用模型并行（例如跨多个核心分割权重和张量）将模型扩展到数十亿个参数 (Shazeer 等人, 2018; Rajbhandari 等人, 2019; Raïl 等人, 2019; Brown 等人., 2020; Shoeybi 等人, 2019）。或者，Harlap 等人。 (2018)；黄等人。 (2019) 建议使用基于管道的模型并行，其中不同的层跨设备分割，微批次通过管道传输到不同的层。最后，提出了 Product Key 网络 (Lample 等人, 2019)，通过根据给定层的传入令牌表示查找可学习嵌入来扩展神经网络的容量。

我们的工作研究了一类进行条件计算的方法中的特定模型，其中计算决策是根据输入动态做出的。Cho 和 Bengio (2014) 提出根据模型隐藏状态中出现的某些位模式自适应地选择权重。艾根等人。 (2013) 使用密集矩阵乘法和 ReLU 激活构建了堆叠专家层，并在抖动的 MNIST 和单调语音上显示了有希望的结果。在计算机视觉领域，Puigcerver 等人。 (2020) 在上游预训练期间根据语义类别手动路由标记，然后根据下游任务选择要使用的相关专家。

开关变压器

Shazeer 等人证明了现代深度学习架构背景下的专家混合 (MoE) 是有效的。 (2017) 该工作添加了一个 MoE 层，该层堆叠在 LSTM (Hochreiter 和 Schmidhuber, 1997) 层之间，并且令牌分别路由到专家组。这在语言建模和机器翻译基准方面产生了最先进的结果。 MoE 层由 Mesh Tensorow 库重新引入到 Transformer 架构中 (Shazeer 等人, 2018)，其中引入 MoE 层作为 FFN 层的替代品，但是没有附带的 NLP 结果。最近，通过机器学习基础设施的进步，GShard (Lepikhin 等人, 2020) 扩展了 XLA 编译器，使用 MoE Transformer 显着改进了 100 种语言的机器翻译。最后范等人。 (2021) 选择了一种不同的确定性 MoE 策略将模型参数分成不重叠的语言组。

Transformer 注意力模式中沿序列长度维度 (L) 的稀疏性已成为一种成功的技术，可以降低 $O(L^2)$ 的注意力复杂度 (Child 等人, 2019 年; Correia 等人, 2019 年; Sukhbaatar 等人, 2019; Kitaev 等人, 2020; Zaheer 等人, 2020; Beltagy 等人, 2020)。这使得能够学习比以前更长的序列。此版本的 Switch Transformer 不采用注意力稀疏性，但这些技术是互补的，并且作为未来的工作，可以将这些技术结合起来，以潜在地改善需要长上下文的任务的学习。

7. 讨论

我们提出并讨论有关 Switch Transformer 和稀疏专家模型的问题，其中稀疏性指的是权重，而不是注意力模式。

由于参数数量众多，开关变压器不是更好吗？是的，而且是设计使然！参数与所使用的总 FLOP 无关，是缩放神经语言模型的有用轴。大型模型已被详尽地证明具有更好的性能 (Kaplan 等人, 2020)。但在这种情况下，我们的模型在使用相同的计算资源时样本效率更高、速度更快。我无法使用超级计算机，这对我还有用吗？尽管这项工作主要集中在非常大的模型上，但我们还发现，只有两名专家的模型可以提高性能，同时轻松满足常用 GPU 或 TPU 的内存限制（详细信息请参阅附录 D）。因此，我们相信我们的技术在所有规模的环境中都是有用的。在速度精度帕累托曲线上，稀疏模型是否优于密集模型？是的。在各种不同的模型大小中，稀疏模型在每步和挂钟时间上都优于密集模型。我们的对照实验表明，在固定的计算量和时间下，稀疏模型优于密集模型。我无法部署万亿参数模型，我们可以缩小这些模型吗？我们无法完全保持模型质量，但通过将稀疏模型提炼为密集模型，可以实现 10 到 100 倍的压缩率，同时实现专家模型质量增益的 30%。为什么使用 Switch Transformer 而不是模型并行密集模型？从时间角度来看，Switch Transformer 比具有分片参数的密集模型要高效得多（图 6）。此外，我们指出，这一决定并不相互排斥——我们

可以并且确实在 Switch Transformer 中使用模型并行，增加了每个代币的 FLOP，但会导致传统模型并行的速度减慢。

为什么稀疏模型还没有被广泛使用？尝试稀疏模型的动机因扩展密集模型的巨大成功而受到阻碍（正如 Hooker (2020) 中所述，密集模型的成功部分是由与深度学习硬件的共同适应驱动的）。此外，稀疏模型还存在多个问题，包括（1）模型复杂性、（2）训练难度和（3）通信成本。Switch Transformer 在缓解这些问题方面取得了长足的进步。

8. 未来的工作

本文提出了一种简化的架构、改进的训练程序以及对稀疏模型如何扩展的研究。然而，仍然存在许多开放的未来方向，我们在此简要描述：

1. 一个重大挑战是进一步提高最大模型的训练稳定性。虽然我们的稳定性技术对于 Switch-Base、Switch-Large 和 Switch-C 模型有效（没有观察到不稳定），但它们不足以用于 Switch-XXL。我们已经采取了早期措施来稳定这些模型，我们认为这通常对大型模型有用，包括使用正则化器来提高稳定性和适应梯度裁剪的形式，但这仍然没有解决。
2. 一般来说，我们发现改进的预训练质量会带来更好的下游结果（附录 E），尽管我们有时会遇到明显的异常情况。例如，尽管对 C4 数据集建模存在类似的困惑，但 1.6T 参数 Switch-C 在 SQuAD 中仅获得 87.7 的精确匹配分数，这与所有 Switch-XXL 模型的 89.6 相比较差。一个显着的区别是，Switch-XXL 模型的每个代币的 FLOPS 是 Switch-C 模型的 10 倍，尽管它的唯一参数少了 4 倍（395B 与 1.6T）。这表明我们对微调质量、每个令牌的 FLOPS 和参数数量之间的依赖关系知之甚少。
3. 对尺度关系进行全面研究，以指导混合数据、模型和专家并行的架构设计。理想情况下，给定硬件配置的规格（计算、内存、通信），人们可以更快地设计最佳模型。反之亦然，这也可能有助于未来硬件的设计。
4. 我们的工作属于自适应计算算法系列。我们的方法始终使用相同、同质的专家，但未来的设计（通过更灵活的基础设施促进）可以支持异构专家。这将能够实现更灵活的适应——当需要更多计算时，通过路由到更大的专家——也许对于更困难的例子。
5. 研究 Transformer FFN 层之外的专家层。我们发现初步证据表明这同样可以提高模型质量。在附录 A 中，我们报告了在 Self-Attention 层中添加这些内容的质量改进，其中我们

层取代了产生 Q、K、V 的权重矩阵。然而，由于 `b6oat16` 格式的训练不稳定，我们将其留作未来工作的领域。

6. Examining Switch Transformer in new and across different modalities. We have thus far only considered language, but we believe that model sparsity can similarly provide advantages in new modalities, as well as multi-modal networks.

这个列表可以很容易地扩展，但我们希望这能让我们了解我们正在考虑的挑战类型以及我们怀疑有希望的未来方向。

9. 结论

Switch Transformer 是可扩展且有效的自然语言学习器。我们简化了专家混合，以产生一种易于理解、训练稳定且比同等大小的密集模型更高的样本效率的架构。我们发现这些模型在各种自然语言任务和不同的训练方案中表现出色，包括预训练、微调和多任务训练。这些进步使得训练具有数千亿到万亿参数的模型成为可能，并且相对于密集的 T5 基线实现了显着的加速。我们希望我们的工作能够激励稀疏模型成为一种有效的架构，并鼓励研究人员和实践者在自然语言任务及其他任务中考虑这些灵活的模型。

Acknowledgments

作者要感谢 Margaret Li，她为算法改进提供了数月的重要见解，并为实证研究提供了建议。Hugo Larochelle 对草案提供了明智的建议和澄清意见，Irwan Bello 提供了详细的评论和仔细的修改，Colin Raïl 和 Adam Roberts 对神经语言模型和 T5 代码库提供了及时的建议，Yoshua Bengio 对研究提供了建议和鼓励在自适应计算领域，Jascha Sohl-dickstein 为稳定新的大规模模型和论文修订提供了有趣的新方向，Google Brain 团队则对论文进行了有用的讨论。Blake Hechtman 在分析和改进我们模型的训练性能方面提供了宝贵的帮助。

A. 切换注意力

沙泽尔等人。（2018）；莱皮欣等人。（2020）通过将 MoE 层添加到 Transformer 的密集前馈网络（FFN）计算中，设计了 MoE Transformer（Shazeer 等人，2017）。同样，我们的工作也替换了 Transformer 中的 FFN 层，但我们在那里简单地探索一种替代设计。我们将 Switch 层添加到 Transformer Self-Attention 层中。为此，我们将生成查询、键和值的可训练权重矩阵替换为 Switch 层，如图 10 所示。

表 10 记录了固定数量的步骤后的质量以及几种变体的训练时间。尽管我们发现了改进，但我们也发现这些层在使用 `b6oat16` 精度时更加不稳定，因此我们没有将它们包含在最终变体中。

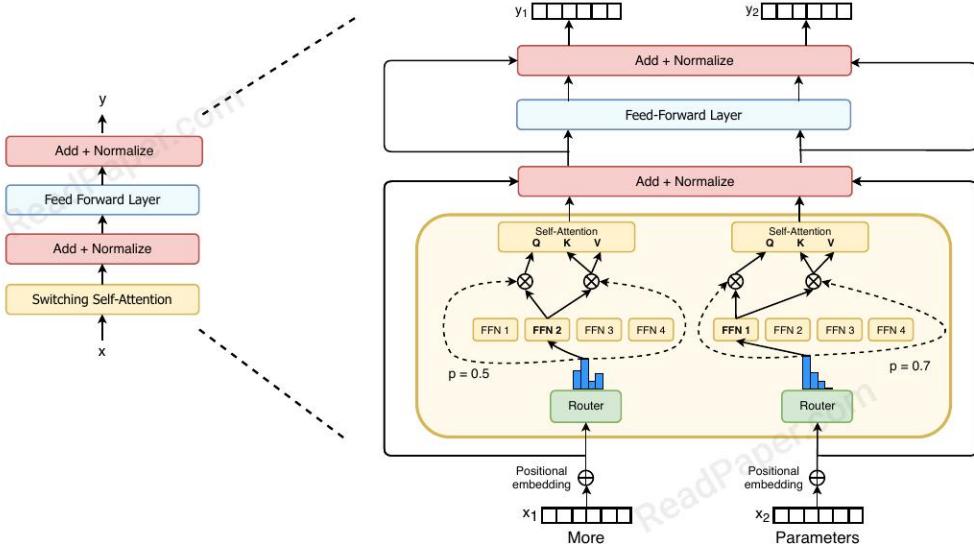


图 10：注意力切换层。我们绘制了如何将 Switch 层合并到自注意力变压器块。对于每个令牌（这里我们显示两个令牌， $x_1 = \hat{a}$ 更多和 $x_2 = \hat{a}$ 参数），一组权重生成查询另一组独特的权重产生共享的键和值。我们在实验中每个专家都是线性操作，以及 FFN，如整个作品都是如此。虽然我们发现使用以下方法可以提高质量我们发现，当使用低精度数字时，它会更加不稳定格式，从而将其留待将来的工作。

然而，当这些层确实稳定地训练时，我们相信初步的积极结果表明了未来有希望的方向。

Model	Precision	Quality @100k Steps (\uparrow)	Quality @16H (\uparrow)	Speed (ex/sec) (\uparrow)
Experts FF	float32	-1.548	-1.614	1480
Expert Attention	float32	-1.524	-1.606	1330
Expert Attention	bfloat16	[diverges]	[diverges]	-
Experts FF + Attention	float32	-1.513	-1.607	1240
Expert FF + Attention	bfloat16	[diverges]	[diverges]	-

表 10：切换注意力层结果。所有模型都有 32 位专家并进行 524k 训练每批次 ken。Experts FF 是专家替换 Transformer 中的 FFN，这是我们整篇论文的标准设置。专家 FF+ 注意的是当使用 Expert 来替换 FFN 和 Self-Attention 层时。当以 bfloat16 精度进行训练时，模型受到了专家的关注发散。

开关变压器

B. 通过 No-Token-Left-Behind 防止令牌丢失

由于 TPU 加速器的软件限制，我们的张量的形状必须是静态大小的。因此，每个专家都有有限且固定的处理令牌表示的能力。然而，这给我们的模型带来了一个问题，该模型在运行时动态路由令牌，可能会导致专家分配不均匀。如果发送给专家的令牌数量小于专家容量，则可以简单地填充计算——硬件的使用效率低下，但在数学上是正确的。然而，当发送给专家的代币数量大于其容量（专家溢出）时，需要一个协议来处理这个问题。莱皮欣等人。（2020）采用了专家混合模型，并通过将其表示传递到下一层来解决专家溢出问题，而无需通过我们也遵循的残差连接进行处理。

我们怀疑不对代币应用计算可能会非常浪费，特别是因为如果一位专家出现溢出，则意味着另一位专家将拥有额外的能力。凭借这种直觉，我们创建了“No-Token-Left-Behind”，它迭代地重新路由最初路由到逾期专家的任何代币。图 11 显示了该方法的图形描述，这将使我们能够保证在训练和推理过程中几乎不会丢失任何令牌。我们假设这可以提高表现并进一步稳定训练，但我们没有发现任何经验上的好处。我们怀疑，一旦网络学习了不同令牌和专家之间的关联，如果这种关联发生变化（例如，将令牌发送给第二高的专家），那么性能可能会下降。

C. 鼓励专家探索

在每个专家层，路由器确定将令牌发送给哪个专家。这是对可用专家的离散决策，以有关代币表示的信息为条件。根据传入的令牌表示，路由器确定最佳专家，但是，它不会收到有关选择替代专家的效果的反事实信息。与强化学习一样，会出现经典的探索-利用困境（Sutton 和 Barto, 2018）。Rosenbaum 等人同样注意到并以不同方式解决了这些问题。（2017）证明了多任务学习的成功。这种特殊的设置与上下文强盗的设置最接近（Robbins, 1952）。确定性地选择顶级专家总是相当于一种剥削策略——我们考虑平衡探索以寻求更好的专家分配。

为了引入探索，我们考虑几种方法：1) 确定性或 argmax 2) 从 softmax 分布中采样 3) 输入表示上的输入丢失 4) 输入表示上的乘性抖动噪声。表 11 报告了由此产生的对模型质量的影响。在整个工作中，我们使用输入抖动来注入噪声，因为我们根据经验发现它的性能最佳。

D. 较低计算状态下的开关变压器

Switch Transformer 在所有规模以及具有数千个核心和数万亿个参数的机制中也是一种有效的架构。我们之前的许多实验都是

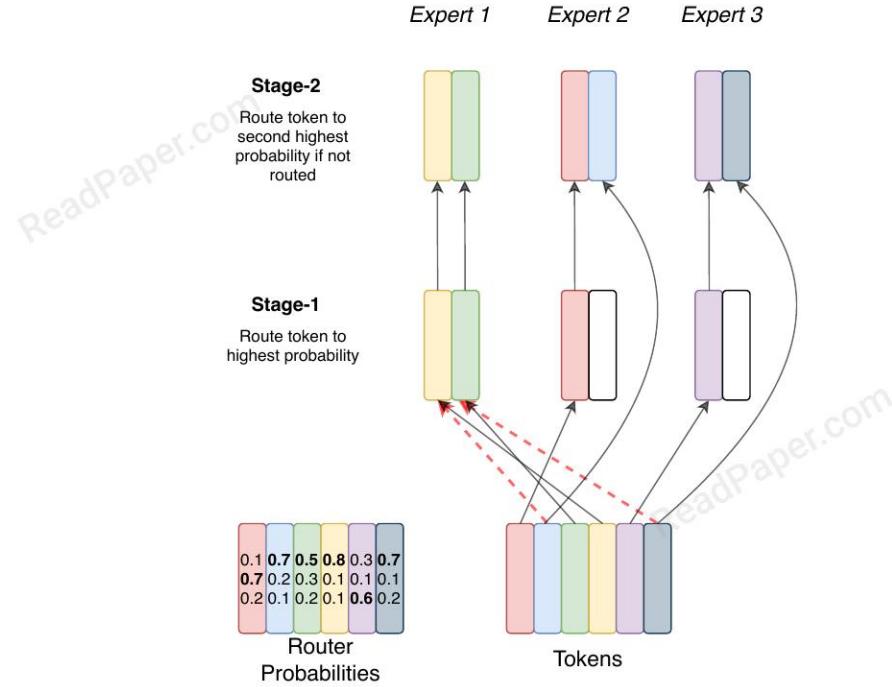


图 11：无令牌遗留路由图。第一阶段相当于Switch
 令牌被路由到概率最高的专家的路由
 路由器。在第 2 阶段，我们查看所有溢出的令牌并路由它们
 到具有第二高概率的专家。代币还是可以的
 如果他们的第二高专家拥有太多代币，则会溢出，但这允许
 大多数要路由的令牌。这个过程可以迭代以保证
 几乎没有任何代币被丢弃。

Model	Quality (Neg. Log Perp.) (\uparrow)
Argmax	-1.471
Sample softmax	-1.570
Input dropout	-1.480
Input jitter	-1.468

表 11：路由器探索策略。开关变压器的质量，通过测量
 负对数困惑度，在不同的随机性选择策略下
 专家（越低越好）。材料速度性能没有差异
 变体之间。

在 10B+ 参数模型的规模上，但我们在图 12 中显示，只要 2 个专家就能比 FLOP 匹配的对应模型产生引人注目的收益。即使超级计算机不易获得，使用 2、4 或 8 名专家（我们通常建议每个核心一名专家）训练 Switch Transformer 也会比 T5 稠密基线带来显著改进。

开关变压器

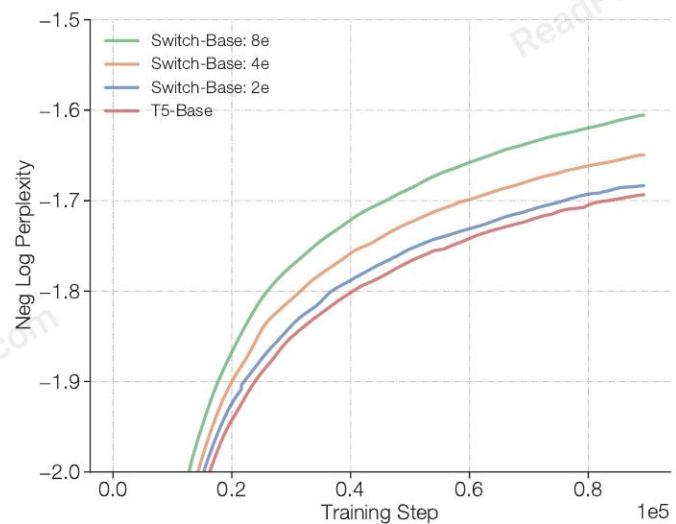


图 12：只有少数专家的开关变压器。开关变压器改进了即使专家很少。在这里，我们展示了非常大的缩放属性所有规模，我们使用 2、4 和 8 个专家改进了 T5-Base 模型。

E. 上游与下游模型性能的关系

无法保证模型在预训练目标上的质量将转化为下游任务结果。图 13 显示了密集模型和 Switch 模型在 C4 预训练任务上的上游模型质量与两个下游任务测量的相关性：平均 SuperGLUE 性能和 TriviaQA 得分。我们选择这两项任务，其中一项任务探索模型的推理，另一项任务探索事实知识。

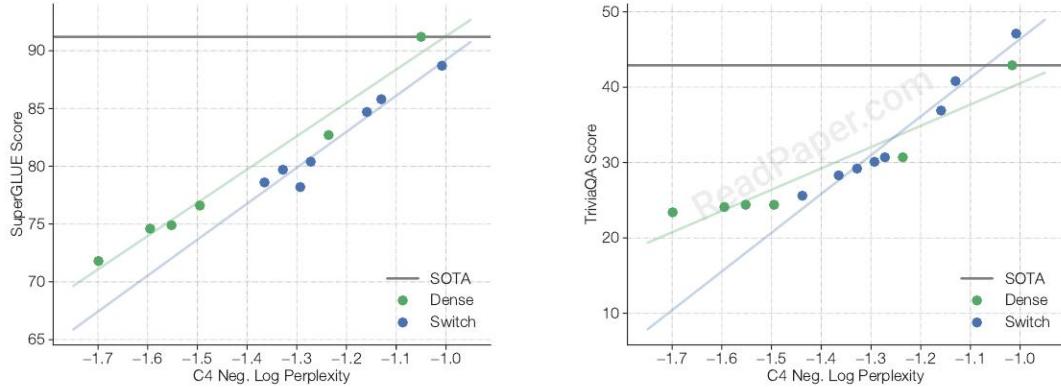


图 13：上游预训练质量到下游模型质量。我们将 SuperGLUE 和 Triv 上的上游性能和下游质量 iaQA（没有SSM记录的SOTA），推理和知识密集型基准- 分别标记（验证集）。我们发现，与基线一样， Switch 模型随着上游预训练任务的改进而扩展。为了 SuperGLUE，我们发现负对数困惑度之间存在松散的线性关系 以及 SuperGLUE 平均得分。然而，密集模型通常表现 更适合固定的困惑，特别是在大规模制度中。反过来，在知识密集型任务 TriviaQA 上，我们发现 Switch Transformer 对于给定的上游困惑度，可以遵循改进的缩放关系， 它比密集的对应物做得更好。进一步的统计数据（收集成本昂贵 并留待未来的工作）有必要证实这些观察结果。

我们发现了一致的相关性，表明对于基线模型和 Switch 模型，改进的预训练会带来更好的下游结果。此外，对于固定的上游困惑度，我们发现 Switch 模型和密集模型在所有到中等模型大小的情况下表现相似。然而，在最大的模型体系 (T5-11B/T5-XXL) 中，我们最大的 Switch 模型 (如第 5.6 节所述) 并不总是能够很好地将其上游困惑转化为下游对 SuperGLUE 任务的微调。这值得未来的调查和研究以充分发挥稀疏模型的潜力。理解专家模型的微调动态非常复杂，并且依赖于正则化、负载平衡和微调参数。

开关变压器

F. 开关变压器的伪代码

网格张量中开关变压器的伪代码 (Shazeer 等人, 2018)。下面的代码没有使用模型并行 (更多详细信息请参见 5.4)。

```

import mesh_tensorflow as mtf

def load_balance_loss(router_probs, expert_mask):
    """Calculate load-balancing loss to ensure diverse expert routing."""
    # router_probs is the probability assigned for each expert per token.
    # router_probs shape: [num_cores, tokens_per_core, num_experts]
    # expert_index contains the expert with the highest router probability in one-hot format.
    # expert_mask shape: [num_cores, tokens_per_core, num_experts]

    # For each core, get the fraction of tokens routed to each expert.
    # density_1 shape: [num_cores, num_experts]
    density_1 = mtf.reduce_mean(expert_mask, reduced_dim=tokens_per_core)

    # For each core, get fraction of probability mass assigned to each expert
    # from the router across all tokens.
    # density_1_proxy shape: [num_cores, num_experts]
    density_1_proxy = mtf.reduce_mean(router_probs, reduced_dim=tokens_per_core)

    # density_1 for a single core: vector of length num_experts that sums to 1.
    # density_1_proxy for a single core: vector of length num_experts that sums to 1.
    # Want both vectors to have uniform allocation (1/num_experts) across all num_expert elements.
    # The two vectors will be pushed towards uniform allocation when the dot product is minimized.
    loss = mtf.reduce_mean(density_1_proxy * density_1) * (num_experts ** 2)
    return loss

```

图 14: Mesh Tensorflow 中 Switch Transformers 负载平衡损耗的伪代码。

```

import mesh_tensorflow as mtf

def router(inputs, capacity_factor):
    """Produce the combine and dispatch tensors used for sending and
    receiving tokens from their highest probability expert. """
    # Core layout is split across num_cores for all tensors and operations.
    # inputs shape: [num_cores, tokens_per_core, d_model]

    router_weights = mtf.Variable(shape=[d_model, numExperts])

    # router_logits shape: [num_cores, tokens_per_core, numExperts]
    router_logits = mtf.einsum([inputs, router_weights], reduced_dim=d_model)

    if is_training:
        # Add noise for exploration across experts.
        router_logits += mtf.random_uniform(shape=router_logits.shape, minval=1-eps, maxval=1+eps)

    # Convert input to softmax operation from bfloat16 to float32 for stability.
    router_logits = mtf.to_float32(router_logits)

    # Probabilities for each token of what expert it should be sent to.
    router_probs = mtf.softmax(router_logits, axis=-1)

    # Get the top-1 expert for each token. expert_gate is the top-1 probability
    # from the router for each token. expert_index is what expert each token
    # is going to be routed to.
    # expert_gate shape: [num_cores, tokens_per_core]
    # expert_index shape: [num_cores, tokens_per_core]
    expert_gate, expert_index = mtf.top_1(router_probs, reduced_dim=numExperts)

    # expert_mask shape: [num_cores, tokens_per_core, numExperts]
    expert_mask = mtf.one_hot(expert_index, dimension=numExperts)

    # Compute load balancing loss.
    aux_loss = load_balance_loss(router_probs, expert_mask)

    # Experts have a fixed capacity, ensure we do not exceed it. Construct
    # the batch indices, to each expert, with position_in_expert
    # make sure that not more than expert_capacity examples can be routed to
    # each expert.
    position_in_expert = mtf.cumsum(expert_mask, dimension=tokens_per_core) * expert_mask

    # Keep only tokens that fit within expert_capacity.
    expert_mask *= mtf.less(position_in_expert, expert_capacity)
    expert_mask_flat = mtf.reduce_sum(expert_mask, reduced_dim=experts_dim)

    # Mask out the experts that have overflowed the expert capacity.
    expert_gate *= expert_mask_flat

    # combine_tensor used for combining expert outputs and scaling with router probability.
    # combine_tensor shape: [num_cores, tokens_per_core, numExperts, expert_capacity]
    combine_tensor = (
        expert_gate * expert_mask_flat *
        mtf.one_hot(expert_index, dimension=numExperts) *
        mtf.one_hot(position_in_expert, dimension=expert_capacity))

    # Cast back outputs to bfloat16 for the rest of the layer.
    combine_tensor = mtf.to_bfloat16(combine_tensor)

    # Create binary dispatch tensor that is 1 if the token gets routed to the corresponding expert.
    # dispatch_tensor shape: [num_cores, tokens_per_core, numExperts, expert_capacity]
    dispatch_tensor = mtf.cast(combine_tensor, tf.bool)

    return dispatch_tensor, combine_tensor, aux_loss

```

图 15: Mesh Tensorow 中 Switch Transformer 的路由器的伪代码。

开关变压器

```
import mesh_tensorflow as mtf

def switch_layer(inputs, n, capacity_factor, num_experts):
    """Distributed switch transformer feed-forward layer."""
    # num_cores (n) = total cores for training the model (scalar).
    # d_model = model hidden size (scalar).
    # num_experts = total number of experts.
    # capacity_factor = extra buffer for each expert.
    # inputs shape: [batch, seq_len, d_model]
    batch, seq_len, d_model = inputs.get_shape()

    # Each core will route tokens_per_core tokens to the correct experts.
    tokens_per_core = batch * seq_len / num_cores

    # Each expert will have shape [num_cores, expert_capacity, d_model].
    # Each core is responsible for sending expert_capacity tokens
    # to each expert.
    expert_capacity = tokens_per_core * capacity_factor / num_experts

    # Reshape to setup per core expert dispatching.
    # shape: [batch, seq_len, d_model] -> [num_cores, tokens_per_core, d_model]
    # Core layout: [n, 1, 1] -> [n, 1, 1]
    inputs = mtf.reshape(inputs, [num_cores, tokens_per_core, d_model])

    # Core Layout: [n, 1, 1] -> [n, 1, 1, 1], [n, 1, 1, 1]
    # dispatch_tensor (boolean) shape: [num_cores, tokens_per_core, num_experts, expert_capacity]
    # dispatch_tensor is used for routing tokens to the correct expert.
    # combine_tensor (float) shape: [num_cores, tokens_per_core, num_experts, expert_capacity]
    # combine_tensor used for combining expert outputs and scaling with router
    # probability.
    dispatch_tensor, combine_tensor, aux_loss = router(inputs, expert_capacity)

    # Matmul with large boolean tensor to assign tokens to the correct expert.
    # Core Layout: [n, 1, 1], -> [1, n, 1, 1]
    # expert_inputs shape: [num_experts, num_cores, expert_capacity, d_model]
    expert_inputs = mtf.einsum([inputs, dispatch_tensor], reduce_dims=[tokens_per_core])

    # All-to-All communication. Cores split across num_cores and now we want to split
    # across num_experts. This sends tokens, routed locally, to the correct expert now
    # split across different cores.
    # Core layout: [1, n, 1, 1] -> [n, 1, 1, 1]
    expert_inputs = mtf.reshape(expert_inputs, [num_experts, num_cores, expert_capacity, d_model])

    # Standard feed forward computation, where each expert will have its own
    # unique set of parameters.
    # Total unique parameters created: num_experts * (d_model * d_ff * 2).
    # expert_outputs shape: [num_experts, num_cores, expert_capacity, d_model]
    expert_outputs = feed_forward(expert_inputs)

    # All-to-All communication. Cores are currently split across the experts
    # dimension, which needs to be switched back to being split across num_cores.
    # Core Layout: [n, 1, 1, 1] -> [1, n, 1, 1]
    expert_outputs = mtf.reshape(expert_outputs, [num_experts, num_cores, expert_capacity, d_model])

    # Convert back to input shape and multiply outputs of experts by the routing probability.
    # expert_outputs shape: [num_experts, num_cores, tokens_per_core, d_model]
    # expert_outputs_combined shape: [num_cores, tokens_per_core, d_model]
    # Core Layout: [1, n, 1, 1] -> [n, 1, 1]
    expert_outputs_combined = mtf.einsum([expert_outputs, combine_tensor], reduce_dims=[tokens_per_core])

    # Remove tokens_per_core shapes used for local routing dispatching to match input shape.
    # Core Layout: [n, 1, 1] -> [n, 1, 1]
    outputs = mtf.reshape(expert_outputs_combined, [batch, seq_len, d_model])
    return outputs, aux_loss
```

图 16: Mesh Tensorow 中 Switch Transformer 层的伪代码。

References

- Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.
- Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1533–1544, 2013.
- Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
- Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.
- Kyunghyun Cho and Yoshua Bengio. Exponentially increasing the capacity-to-computation ratio for conditional computation in deep learning. *arXiv preprint arXiv:1406.7362*, 2014.
- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.
- Gonçalo M Correia, Vlad Niculae, and André FT Martins. Adaptively sparse transformers. *arXiv preprint arXiv:1909.00015*, 2019.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- David Eigen, Marc’Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013.
- Angela Fan, Shruti Bhosale, Holger Schwenk, Zhiyi Ma, Ahmed El-Kishky, Siddharth Goyal, Mandeep Baines, Onur Celebi, Guillaume Wenzek, Vishrav Chaudhary, et al. Beyond english-centric multilingual machine translation. *Journal of Machine Learning Research*, 22(107):1–48, 2021.
- William Fedus, Ian Goodfellow, and Andrew M Dai. Maskgan: Better text generation via filling in the_. *arXiv preprint arXiv:1801.07736*, 2018.
- Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning. *arXiv preprint arXiv:2006.10901*, 2020.
- Scott Gray, Alec Radford, and Diederik P Kingma. Gpu kernels for block-sparse weights. <https://openai.com/blog/block-sparse-gpu-kernels/>, 2017.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*, 2020.

Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018.

卡尔·莫里茨·赫尔曼、托马斯·科西斯基、爱德华·格雷芬斯特、莱塞·埃斯佩霍特、威尔·凯、穆斯塔法·苏莱曼和菲尔·布伦瑟姆。教机器阅读和理解。见 C. Cortes、N. Lawrence、D. Lee、M. Sugiyama 和 R. Garnett，编辑，《神经信息处理系统进展》，第 28 卷，第 13â 1701 页。Curran Associates, Inc., 2015 年。网址 <https://proceedings.neurips.cc/paper/2015/file/afdec7005cc9f14302cd0474fd0f3c96-Paper.pdf>。

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sara Hooker. The hardware lottery. *arXiv preprint arXiv:2009.06489*, 2020.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in neural information processing systems*, pages 103–112, 2019.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

汤姆·科维亚特科斯基 (Tom Kwiatkowski)、詹尼玛利亚·帕洛马基 (Jennimaria Palomaki)、奥利维亚·雷德 (Olivia Redfield)、迈克尔·柯林斯 (Michael Collins)、安库尔·帕里克 (Ankur Parikh)、克里斯·阿尔贝蒂 (Chris Alberti)、丹尼尔·爱泼斯坦 (Danielle Epstein)、伊利亚·波洛苏欣 (Ilia Polosukhin)、雅各布·德夫林 (Jacob Devlin)、肯顿·李 (Kenton Lee) 等。自然语言处理研究的基准。计算语言学协会汇刊, 7:453â 466, 2019。

- Guillaume Lample, Alexandre Sablayrolles, Marc’Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. In *Advances in Neural Information Processing Systems*, pages 8548–8559, 2019.
- Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. *arXiv preprint arXiv:2107.06499*, 2021.
- Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.
- Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.
- Shashi Narayan, Shay B Cohen, and Mirella Lapata. Don’t give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*, 2018.
- Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. Adversarial nli: A new benchmark for natural language understanding. *arXiv preprint arXiv:1910.14599*, 2019.
- Joan Puigcerver, Carlos Riquelme, Basil Mustafa, Cedric Renggli, André Susano Pinto, Sylvain Gelly, Daniel Keysers, and Neil Houlsby. Scalable transfer learning with expert models. *arXiv preprint arXiv:2009.13239*, 2020.
- Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.
- Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimization towards training a trillion parameter models. *arXiv preprint arXiv:1910.02054*, 2019.
- Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.
- Prajit Ramachandran and Quoc V Le. Diversity and depth in per-example routing models. In *International Conference on Learning Representations*, 2018.
- Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

开关变压器

Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? *arXiv preprint arXiv:2002.08910*, 2020.

Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*, 2017.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740, 2020.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2019.

Noam Shazeer. Glu variants improve transformer, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-tensorflow: Deep learning for supercomputers. In *Advances in Neural Information Processing Systems*, pages 10414–10423, 2018.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

尼蒂什·斯里瓦斯塔瓦 (Nitish Srivastava)、杰瑞·E·辛顿 (Geoffrey Hinton) Dropout: 一种防止神经网络过度拟合的简单方法。机器学习研究杂志, 15(1): 1929–1958, 2014。URL <http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>。

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers. *arXiv preprint arXiv:1905.07799*, 2019.

Rich Sutton. The Bitter Lesson. <http://www.incompleteideas.net/IncIdeas/BitterLesson.html>, 2019.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Stanford University, 2018.

Wilson L Taylor. “cloze procedure”: A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433, 1953.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Alex Wang、Yada Pruksachatkun、Nikita Nangia、Amanpreet Singh、Julian Michael、Felix Hill、Omer Levy 和 Samuel Bowman。SuperGlue：通用语言理解系统的更具粘性的基准。《神经信息处理系统进展》，第 3266â 3280 页，2019 年。

Shibo Wang and Pankaj Kanwar. Bfloat16: The secret to high performance on cloud tpus. *Google Cloud Blog*, 2019.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mt5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934*, 2020.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2020.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020.