MoE经典论文简读

参考资料：

- MoE (Mixture-of-Experts) 经典文章简读
- Mixture-of-Experts (MoE) 经典论文一览

1.开创工作

1.1 Adaptive mixtures of local experts, Neural Computation'1991

- 期刊/会议：Neural Computation (1991)
- 论文链接：https://readpaper.com/paper/2150884987
- 代表性作者：Michael Jordan, Geoffrey Hinton

这是大多数 MoE 论文都引用的最早的一篇文章，发表于 1991 年，作者中有两个大家熟知的大佬：Michael Jordan 和 Geoffrey Hinton。

提出了一种新的监督学习过程，一个系统中包含多个分开的网络，每个网络去处理全部训练样本的一个子集。这种方式可以看做是把多层网络进行了模块化的转换。

假设我们已经知道数据集中存在一些天然的子集（比如来自不同的 domain，不同的 topic），那么用单个模型去学习，就会受到很多干扰（interference），导致学习很慢、泛化困难。这时，我们可以使用多个模型（即专家，expert）去学习，使用一个门网络（gating network）来决定每个数据应该被哪个模型去训练，这样就可以减轻不同类型样本之间的干扰。

其实这种做法，也不是该论文第一次提出的，更早就有人提出过类似的方法。对于一个样本 c，第 i 个 expert 的输出为 o_i^c，理想的输出是 d^c，那么损失函数就这么计算：

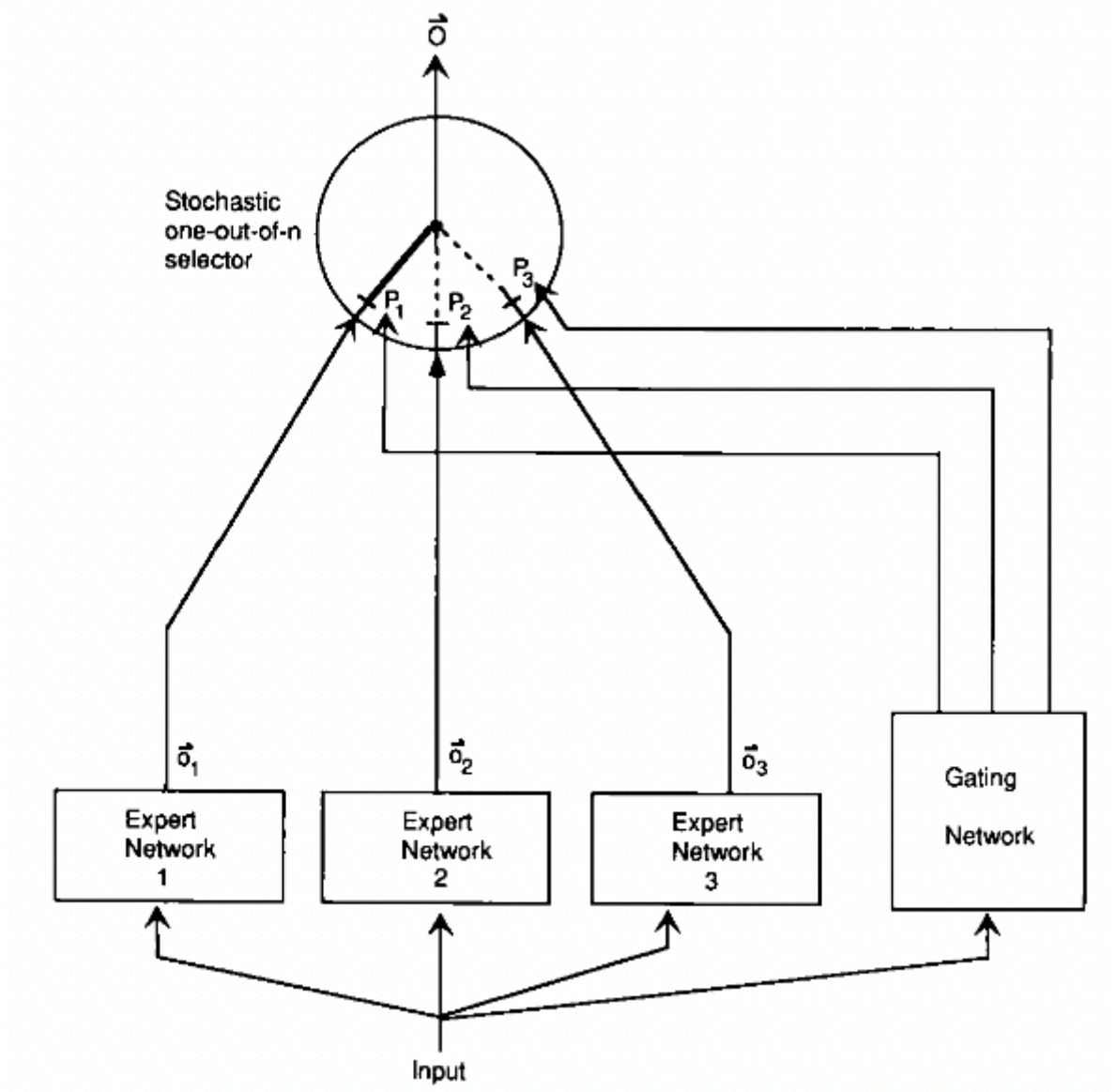
$$E^c = \left\| d^c - \sum_i p_i^c o_i^c \right\|^2$$

其中 p_i^c 是 gating network 分配给每个 expert 的权重，相当于多个 expert 齐心协力来得到当前样本 c 的输出。

这是一个很自然的设计方式，但是存在一个问题：**因为每个 expert 的梯度都受到整体误差的影响，并且它们的贡献大小由 gating 权重决定。**所以不同的 expert 之间的互相影响会非常大，一个 expert 的参数改变了，其他的都会跟着改变，即所谓牵一发而动全身。这样的设计，最终的结果就是一个样本会使用很多的 expert 来处理。于是，这篇文章设计了一种新的方式，调整了一下 loss 的设计，来鼓励不同的 expert 之间进行竞争：

$$E^c = \sum_i p_i^c \|d^c - o_i^c\|^2$$

就是让不同的 expert 单独计算 loss，然后在加权求和得到总体的 loss。这样的话，每个专家，都有独立判断的能力，而不用依靠其他的 expert 来一起得到预测结果。对**每个 expert 的预测单独计算误差**，然后按 gating 权重加权求和。
下面是一个示意图：



在这种设计下，我们将 experts 和 gating network 一起进行训练，最终的系统就会倾向于让一个 expert 去处理一个样本。

上面的两个 loss function，其实长得非常像，但是一个是鼓励合作，一个是鼓励竞争。这一点还是挺启发人的。

论文还提到另外一个很启发人的 trick，就是上面那个损失函数，作者在实际做实验的时候，用了一个变体，使得效果更好：

$$\text{Original}: E^c = \sum_i p_i^c \|d^c - o_i^c\|^2$$

$$\text{Modified}: E^c = -\log \sum_i p_i^c e^{-\frac{1}{2}\|d^c - o_i^c\|^2}$$

对比一下可以看出，在计算每个 expert 的损失之后，先把它给指数化了再进行加权求和，最后取了 log。这也是一个我们在论文中经常见到的技巧。这样做有什么好处呢，我们可以对比一下二者在反向传播的时候有什么样的效果，使用 E^c 对第 i 个 expert 的输出求导，分别得到：

$$\text{original derivative}: \frac{\partial E^c}{\partial o_i^c} = -2p_i^c(d^c - o_i^c)$$

$$\text{new derivative}: \frac{\partial E^c}{\partial o_i^c} = -\left[\frac{p_i^c e^{-\frac{1}{2}\|d^c - o_i^c\|^2}}{\sum_j p_j^c e^{-\frac{1}{2}\|d^c - o_j^c\|^2}}\right](d^c - o_i^c)$$

这个导数可以拆解成三部分：

- 当前 expert 的误差项：$\left(d^c - o_i^c\right)$
- 当前 expert 的门控权重：p_i^c
- 当前 expert 相对于所有 expert 的“相对置信度”：

$$\frac{e^{-\frac{1}{2}\left\|d^c - o_i^c\right\|^2}}{\sum_j e^{-\frac{1}{2}\left\|d^c - o_j^c\right\|^2}}$$

也就是说，一个 expert 的梯度不仅取决于自己的误差，还取决于它与其他 expert 的相对表现。

可**以看到，前者的导数，只会跟当前 expert 有关，但后者则还考虑其他 experts 跟当前 sample c 的匹配程度。换句话说，如果当前 sample 跟其他的 experts 也比较匹配，那么 E^c 对第 i 个 expert 的输出的导数也会相对更小一些。**（其实看这个公式，跟我们现在遍地的对比学习 loss 真的很像！很多道理都是相通的）

1.2 Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer, ICLR'17

- 期刊/会议：ICLR'17
- 论文链接：https://readpaper.com/paper/2952339051
- 代表性作者：Quoc Le, Geoffrey Hinton, Jeff Dean

在 2010 至 2015 年间，两个独立的研究领域为混合专家模型 (MoE) 的后续发展做出了显著贡献：

1. **组件专家**：在传统的 MoE 设置中，整个系统由一个门控网络和多个专家组成。在支持向量机 (SVMs) 、高斯过程和其他方法的研究中，MoE 通常被视为整个模型的一部分。然而，Eigen、Ranzato 和 Ilya 的研究 探索了将 MoE 作为更深层网络的一个组件。这种方法**允许将 MoE 嵌入到多层网络中的某一层，使得模型既大又高效。**
2. **条件计算（Conditional Computation）**：传统的神经网络通过每一层处理所有输入数据。在这一时期，Yoshua Bengio 等研究人员开始探索**基于输入 token 动态激活或停用网络组件**的方法。

传统的神经网络中，每个输入都要经过每一层的所有神经元，所以模型越大，计算量就越高，推理速度就越慢。而 **MoE 的核心思想是：对于不同的输入，只激活一小部分“专家”网络**，其余专家不参与计算。这样就可以构建非常大的模型，但每次推理时实际运行的参数却很少。在 2017 年，Shazeer 等人将这一概念应用于 137B 的 LSTM 。通过引入稀疏性，这项工作在保持极高规模的同时实现了快速的推理速度。在牺牲极少的计算效率的情况下，把模型规模提升**1000多倍。**

这篇文章，从title上就可以看出来它的背景和目的——希望做出极大的神经网络。在此之前，有很多**conditional computational** 的工作，在理论上可以在有限的计算成本内把模型做的非常大，但是那些方法在具体实现的时候，有各种各样的问题。这篇文章提出了 Sparsely-Gated Mixture-of-Experts layer ，声称终于解决了传统 conditional computational 的问题，在牺牲极少的计算效率的情况下，把模型规模提升1000多倍。

（1）Sparsely-Gated Mixture-of-Experts layer

跟1991年那个工作对比，这里的MoE主要有两个区别：

- **Sparsely-Gated**：不是所有expert都会起作用，而是极少数的expert会被使用来进行推理。这种稀疏性，也使得我们可以使用海量的experts来把模型容量做的超级大。
- **token-level**：前面那个文章，是 sample-level 的，即不同的样本，使用不同的experts，但是这篇则是 token-level 的，一个句子中不同的token使用不同的experts。

这篇文章是在RNN的结构上加入了MoE layer：

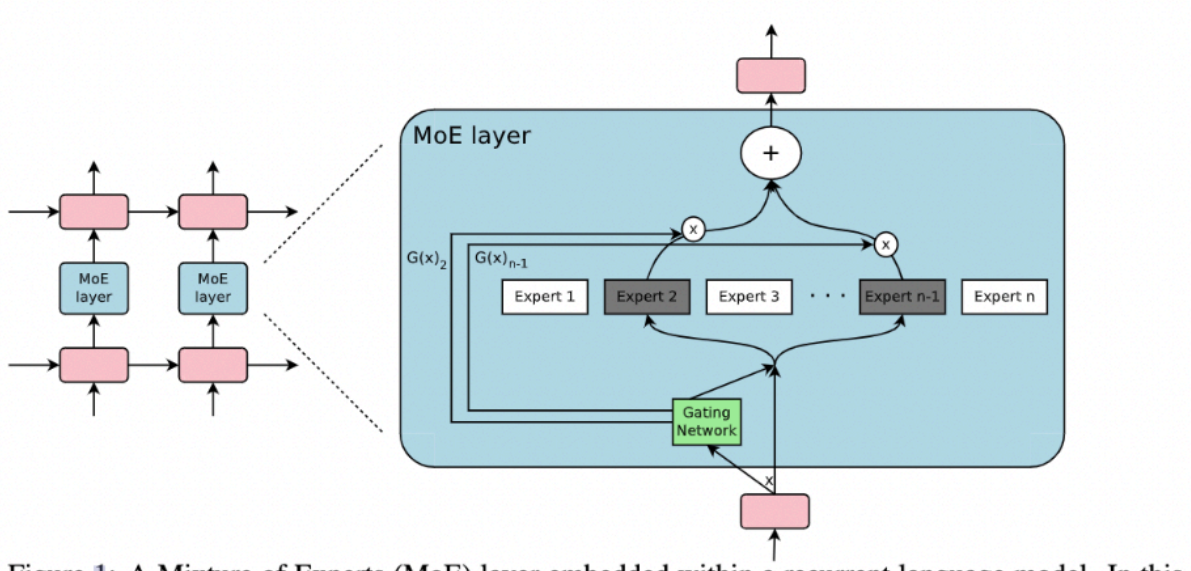


Figure 1: A Mixture of Experts (MoE) layer embedded within a recurrent language model. In this case, the sparse gating function selects two experts to perform computations. Their outputs are modulated by the outputs of the gating network.

如图所示，每个token对应的position，都会有一个MoE Layer，每个MoE layer中包含了一堆的experts，每个expert都是一个小型的FFN，还有一个gating network会根据当前position的输入，选择少数几个expert来进行计算。

(2) Gating Network

设 $G(x)$ 和 $E_i(x)$ 分别是 gating network 和第 i 个 expert 的输出，那么对于在当前 position 的输入 x，输出就是所有 experts 的加权和：

$$y = \sum_{i=1}^{n} G(x)_i E_i(x)$$

(跟第一篇论文的第一个公式类似)

但是这里我们可能有上千个 experts，如果每个都算的话，计算量会非常大，所以这里的一个关键就是希望 $G(x)$ 的输出是稀疏的，只有部分的 experts 的权重是大于 0 的，其余等于 0 的 expert 直接不参与计算。

首先看传统的 gating network 如何设计：

$$G_\sigma(x) = \text{Softmax}(x \cdot W_g)$$

作者提出：

$$G(x) = \text{Softmax}\left(\text{KeepTopK}(H(x), k)\right)$$

其中：

$$\text{KeepTopK}(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ in top K elements of } v \\ -\infty & \text{otherwise} \end{cases}$$

这一步的作用：

- 对于每一个输入 x，先通过某种方式得到一个得分向量 $H(x)$
- 然后只保留前 K 个最高分的元素，其余全部设置为负无穷
- 经过 softmax 后，这些负无穷的位置就变成了 0，对应专家被“关闭”

为了防止某些专家永远不被选中、而另一些专家总是被过度使用，作者在 $H(x)$ 中加入了噪声项：

$$H(x)_i = (x \cdot W_g)_i + \underbrace{\text{StandardNormal}() \cdot \text{Softplus}((x \cdot W_{\text{noise}})_i)}_{\text{Noise term}}$$

分析这个噪声项：

- **StandardNormal()**：从标准正态分布中采样一个随机数（均值为 0，方差为 1）
- **Softplus((x · W_noise)_i)**：将输入线性变换后通过 softplus 函数，确保结果是非负的，作为该专家噪声的标准差
- 所以整体上是一个以 0 为中心、方差由输入决定的噪声项

根据论文原文，实验中通常使用的参数是：

- $K = 2 \sim 4$：即每次只激活 2 到 4 个专家
- $\text{Expert Count} = 100 \sim 1000$：整个 MoE 层可以包含上百甚至上千个专家
- 推理时只激活极小部分专家，因此尽管总参数量巨大，推理速度仍可接受

(3) Expert Balancing

作者在实验中发现，不同 experts 在竞争的过程中，会出现"赢者通吃"的现象：前期变现好的 expert 会更容易被 gating network 选择，导致最终只有少数的几个 experts 真正起作用。因此作者额外增加了一个 loss，来缓解这种不平衡现象，公式如下：

$$\text{Importance}(X) = \sum_{x \in X} G(x)$$

$$L(X) = \lambda \cdot CV(\text{Importance}(X))^2$$

其中 X 代表的是一个 batch 的样本，把一个 batch 所有样本的 gating weights 加起来，然后计算变异系数（coefficient of variation）。总之，这个反映了不同 experts 之间不平衡的程度。最后这个 loss 会加到总体 loss 中，鼓励不同的 experts 都发挥各自的作用。

上面就是 Sparsely-Gated MoE 的主要理论，作者主要在 language modeling 和 machine translation 两个任务上做了实验，因为这两个任务，都是特别受益于大数据和大模型的，而本文的 MoE 的作用主要就在于极大地扩大了模型容量——通过 MoE，把 RNN-based 网络做到了 137B（1.3 千亿）参数的规模，还是挺震撼的。效果自然也是极好的。

经过训练呢，作者发现不同的 experts 确实分化出了不同的"专业"：

Table 9: Contexts corresponding to a few of the 2048 experts in the MoE layer in the encoder portion of the WMT'14 En→ Fr translation model. For each expert i, we sort the inputs in a training batch in decreasing order of $G(x)_i$, and show the words surrounding the corresponding positions in the input sentences.

Expert 381	Expert 752	Expert 2004
... with **researchers** , ...	... plays **a** core ...	... with **rapidly** growing ...
... to **innovation** .	... plays **a** critical ...	... under **static** conditions ...
... tics **researchers** .	... provides **a** legislative ...	... to **swift** ly ...
... the **generation** of ...	... play **a** leading ...	... to **dras** tically ...
... technology **innovations** is ...	... assume **a** leadership ...	... the **rapid** and ...
... technological **innovations** , ...	... plays **a** central ...	... the **fast** est ...
... support **innovation** throughout ...	... taken **a** leading ...	... the **Quick** Method ...
... role **innovation** will ...	... established **a** reconciliation ...	... rec **urrent**) ...
... research **scienti** st ...	... played **a** vital ...	... provides **quick** access ...
... promoting **innovation** where ...	... have **a** central ...	... of **volatile** organic ...
...	...	...

上面的两篇，是MoE系列工作的基础，接下来介绍的工作，都是近几年的比较出名的工作：

2.使用 MoE 开发超大模型

2.1 GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding, ICLR'21

- 期刊/会议：ICLR'21
- 论文链接：https://readpaper.com/paper/3040573126

GShard，按照文章的说法，是第一个将MoE的思想拓展到Transformer上的工作。具体的做法是，把Transformer的encoder和decoder中，**每隔一个FFN层替换成position-wise 的 MoE 层**，使用的都是Top-2 gating network。

每隔一个 FFN 层 替换一次，也就是说：

Layer	Original	GShard
0	FFN	MoE
1	FFN	FFN
2	FFN	MoE
3	FFN	FFN

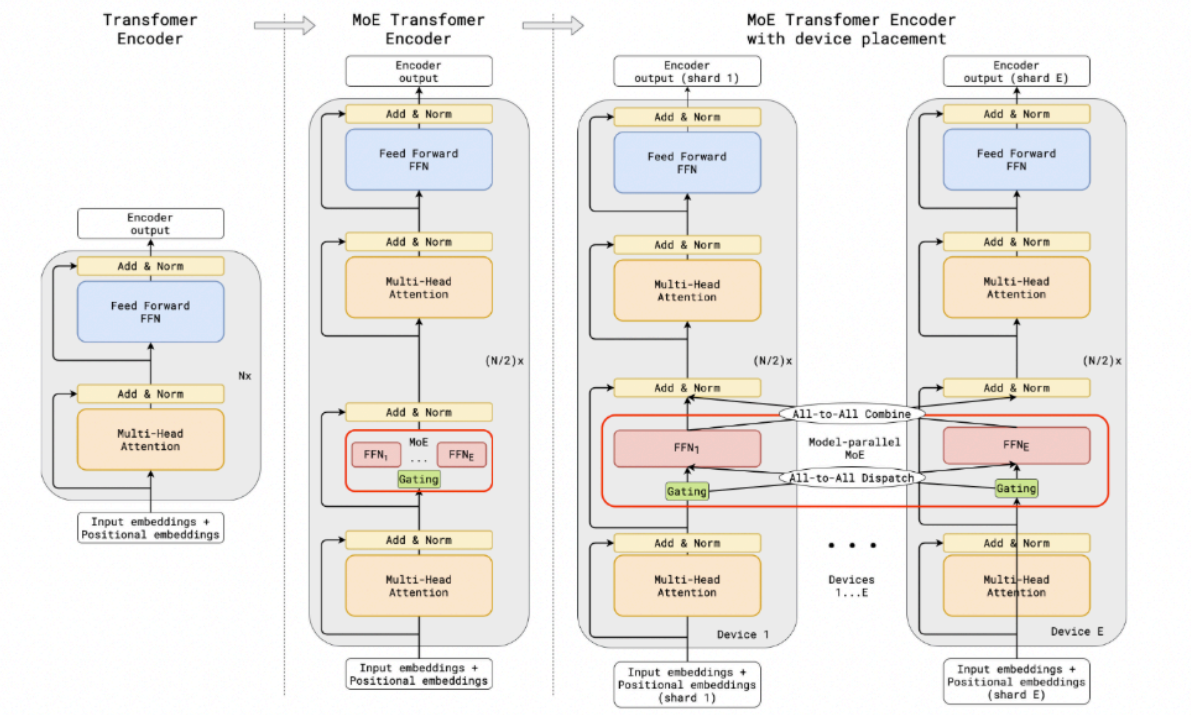


Figure 3: Illustration of scaling of Transformer Encoder with MoE Layers. The MoE layer replaces the every other Transformer feed-forward layer. Decoder modification is similar. (a) The encoder of a standard Transformer model is a stack of self-attention and feed forward layers interleaved with residual connections and layer normalization. (b) By replacing every other feed forward layer with a MoE layer, we get the model structure of the MoE Transformer Encoder. (c) When scaling to multiple devices, the MoE layer is sharded across devices, while all other layers are replicated.

1. **标准 Transformer（a）**：是标准的Transformer编码器，其中每个 token 通过一个标准的 FFN。
2. **MoE Transformer（b）**：将每隔一个的 FFN 层替换为 MoE 层。这意味着在编码器中，不再是每个 token 都通过相同的 FFN，而是通过一个由多个专家组成的 MoE 层。

3. **MoE跨设备分片（c）**：它展示了 MoE 层是如何在多个设备上进行分片的。GShard MoE 层中的**专家网络（experts）被分布在不同的设备上。**每个专家网络负责处理一部分输入数据，并且每个 token 根据门控机制的输出被分配到一个或两个专家网络中。这样，整个 MoE 层的计算被分散到了多个设备上，每个设备负责处理一部分计算任务。将输入数据分配到对应的专家设备，然后将不同设备上的专家输出汇总到最终结果。

实现 **MoE 跨设备分片的关键技术是模型并行化（model parallelism）和数据并行化（data parallelism）的结合。**在模型并行化中，模型的不同部分（在这里是 MoE 层的专家网络）被分配到不同的设备上。在数据并行化中，输入数据（token）被分割成多个部分，每个部分被分配给不同的设备进行处理。

为了实现这种分片，论文中提到的 GShard 模块提供了一套 API 和编译器扩展，允许用户在模型代码中简单地注释关键张量，指定它们应该如何在设备集群上进行分片。这样，编译器就可以自动地将计算图（computation graph）转换为可以在多个设备上并行执行的程序，而不需要用户手动处理复杂的数据分片和通信逻辑。

由于专家被分配到不同设备，可以并行计算，因此大大提升了模型的计算效率，这也解释了为什么 MoE 可以实现更大模型参数、更低训练成本。

为了保持负载平衡和训练效率，GShard 的作者除了引入上节 Sparsely-Gated MoE 中的辅助 loss 外，还引入了一些关键变化：

- **随机路由**：在 Top-2 设置中，GShard 始终选择排名最高的专家，但第二个专家是根据其权重比例随机选择的。
- **专家容量**：可以设定一个阈值，定义一个专家能处理多少 token。如果两个专家的容量都达到上限，token 就会溢出，并通过残差连接传递到下一层，或在某些情况下被完全丢弃。专家容量是 MoE 中最重要的概念之一。

专家容量的作用

原因	说明
1. 静态编译与调度优化	硬件（TPU/GPU）要求编译时知道输入输出张量形状，容量机制使 expert 输入形状固定
2. 控制负载均衡	避免某些 expert 过载，限制最大处理 token 数量
3. 提升训练稳定性	鼓励 gate 学会更均匀地分配 token，避免部分 expert 训练不足

注意：在推理过程中，只有部分专家被激活。同时，有些计算过程是共享的，例如自注意力 (self-attention) 机制，它适用于所有 token。**这就解释了为什么我们可以使用相当于 12B 密集（Dense）模型的计算资源来运行一个包含 8 个专家的 47B 模型。**如果我们采用 Top-2 门控，模型会使用高达 14B 的参数。但是，由于自注意力操作 (专家间共享) 的存在，实际上模型运行时使用的参数数量是 12B。

参数	含义
47B	整个模型的参数总量，包括所有专家和共享层（如 self-attention）
12B	实际在每次推理中使用的参数量（因为只激活了部分专家）

文中还提到了很多其他设计：

- **Expert capacity balancing**：强制每个expert处理的tokens数量在一定范围内。

- **Local group dispatching**：通过把一个batch内所有的tokens分组，来实现并行化计算。将输入 batch 中的 token 划分成多个小组（group），每个小组独立进行 expert 分配和 dispatch（派发），而不是整个 batch 统一调度。
- **Auxiliary loss**：也是为了缓解"赢者通吃"问题。在训练过程中，在主损失函数之外，加入一个额外的损失项，用于优化门控网络的行为。
- **Random routing**：在Top-2 gating的设计下，两个expert如何更高效地进行routing。在 Top-K gating 的基础上，给 expert 的选择过程引入一定的随机性，以提升负载均衡性和模型鲁棒性。在 gate 的 logits 上添加随机噪声，再做 softmax 和 Top-K 选择，这样即使两个 expert 的原始得分相近，也能通过噪声决定谁被选中。

2.2 Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity, JMLR'22

- 期刊/会议：JMLR'22
- 论文链接：https://readpaper.com/paper/4568736324836663297

虽然发表是2022年才在发表在JMLR上，Swith Transformer实际上在21年就提出了。它是在**T5模型的基础上加入了MoE设计**，并在C4数据集上预训练，得到了一个"又快又好"的预训练大模型。

Swith Transformer 的主要亮点在于——**简化了MoE的routing算法，从而大大提高了计算效率。**

结构如下：

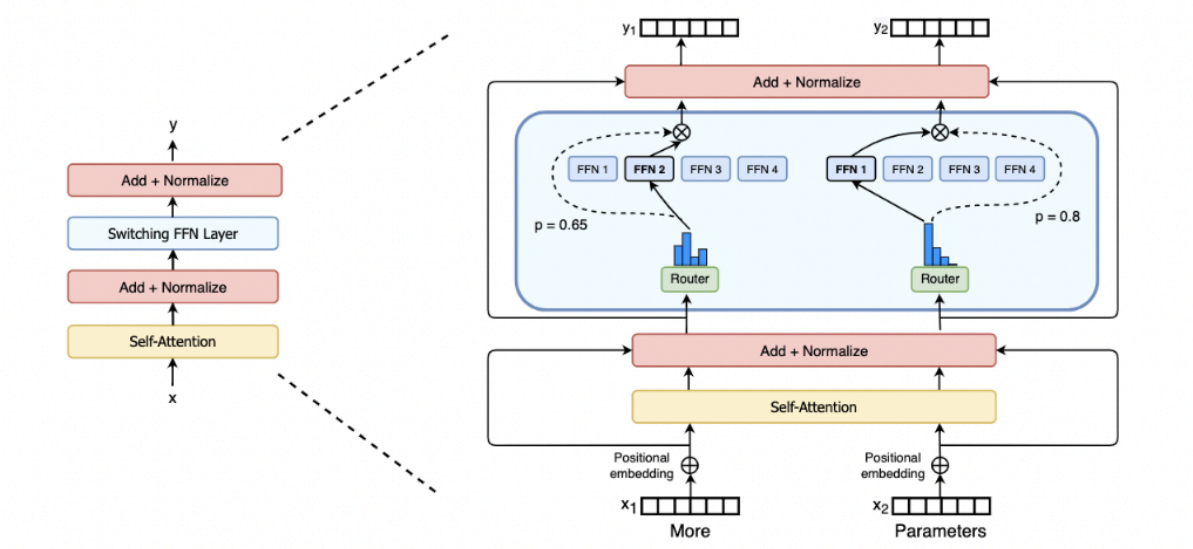


Figure 2: **Illustration of a Switch Transformer encoder block**. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

Swith Transformer 在论文中提到其设计的指导原则是——**尽可能地把Transformer模型的参数量做大!**（同时以一种简单高效的实现方式）。跟其他MoE模型的一个显著不同就是，**Switch Transformer 的 gating network 每次只 route 到 1 个 expert**，而其他的模型都是至少2个。这样就是最稀疏的MoE了，因此单单从MoE layer的计算效率上讲是最高的了。

原始 MoE vs. Switch MoE：

特性	传统 MoE（如 GShard、DeepSeek-MoE）	Switch MoE（Switch Transformer）
专家选择机制	Top-K gating（通常 K=2）	Top-1 gating（只选一个 expert）
输出方式	所有被选中的 expert 权重加权求和	只激活一个 expert，无权重加权
稀疏程度	中等稀疏	最大稀疏（最节省计算）
负载均衡要求	高（多个 expert 共同承担）	更高（每个 token 只能选一个 expert）

下图展示了在同样的计算开销下，增大 experts 个数带来的性能提升，反正就是全面吊打T5，而且效率还一样：

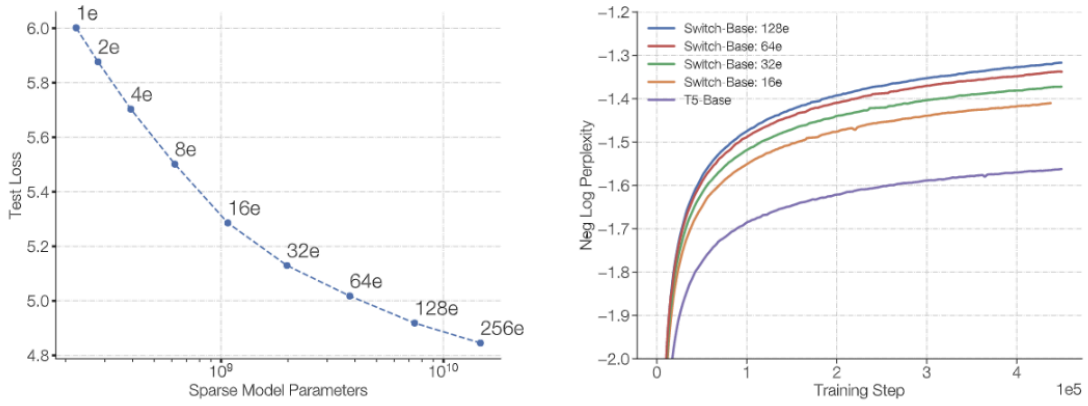


Figure 4: **Scaling properties of the Switch Transformer. Left Plot:** We measure the quality improvement, as measured by perplexity, as the parameters increase by scaling the number of experts. The top-left point corresponds to the T5-Base model with 223M parameters. Moving from top-left to bottom-right, we double the number of experts from 2, 4, 8 and so on until the bottom-right point of a 256 expert model with 14.7B parameters. Despite all models using an equal computational budget, we observe consistent improvements scaling the number of experts. **Right Plot:** Negative log-perplexity per step sweeping over the number of experts. The dense baseline is shown with the purple line and we note improved sample efficiency of our Switch-Base models.

2.3 GLaM: Efficient Scaling of Language Models with Mixture-of-Experts, 2021

- 年份：2021
- 论文链接：https://readpaper.com/paper/4568736324836663297
- Google Blog：https://ai.googleblog.com/2021/12/more-efficient-in-context-learning-with.html

这是Google在2021年推出的一个超大模型，比GPT-3大三倍，但是由于使用了Sparse MoE的设计，训练成本却只有GPT-3的1/3，而且在29个NLP任务上超越了GPT-3。

下面这个来自Google Blog的动图很形象地展示了GLaM的结构：

其实我们可以发现，跟GShard几乎一模一样。尽管 GLaM 基于传统 MoE 结构，但它也做了许多工程和训练上的优化，使其更适合大规模语言模型：

改进点	描述
Expert 数量极大	GLaM 使用了多达 256 ~ 1024 个 expert，总参数量达到**1.2T（万亿）**
分层 MoE 设计	仅在部分 Transformer 层使用 MoE，其余仍用普通 FFN，控制训练难度
Top-2 Gating + Noise	门控网络加入噪声项，防止某些 expert 被过度使用
Auxiliary Loss	添加辅助损失函数，鼓励 gate 分配更均匀
Expert Capacity 控制	引入容量限制机制，防止某些 expert 过载
分布式训练支持	利用 XLA 和 TPU 实现高效分布式调度与通信

Model Name	Model Type	n_{params}	$n_{\text{act-params}}$
BERT	Dense Encoder-only	340M	340M
T5	Dense Encoder-decoder	13B	13B
GPT-3	Dense Decoder-only	175B	175B
Jurassic-1	Dense Decoder-only	178B	178B
Megatron-530B	Dense Decoder-only	530B	530B
GShard-M4	MoE Encoder-decoder	600B	1.5B
Switch-C	MoE Encoder-decoder	1.5T	1.5B
GLaM(64B/64E)	MoE Decoder-only	1.2T	96.6B

上表展示了GLaM跟其他大模型的对比。可以看到，虽然GLaM的总参数量有1.2T，但是在计算式实际激活的参数量只有96B，所以在inference的时候，比GPT-3等dense model要快得多。

GLaM使用的数据量也比Switch-Transformer等要大得多：

Dataset	Tokens (B)	Weight in mixture
Filtered Webpages	143	0.42
Wikipedia	3	0.06
Conversations	174	0.28
Forums	247	0.02
Books	390	0.20
News	650	0.02

Table 3. Data and mixture weights in GLaM training set. During training, we sample from different dataset sources with probability proportional to "weight in mixture". During training, we limit the number of epochs each dataset is seen to prevent the model overfit on smaller datasets.

反正最终的结果，是一个比GPT-3更快更强大的通用LM。

2.4 小结

上面的三篇文章（GShard，Switch-Transformer，GLaM）都是希望通过MoE的方式把模型做得尽可能的大，大到普通人玩不起（动辄使用几百个experts），下面介绍的两篇文章，则更加亲民一点，是关于如何利用MoE去压缩模型、提高效率：

3.使用 MoE 来使模型轻量化

3.1 Go Wider Instead of Deeper, AAAI'22

- 期刊/会议：AAAI'22
- 论文链接：https://readpaper.com/paper/3184020733

这个文章名字比较唬人，思路也比较新颖，所以介绍一下。**与其把模型做深（增加层数），不如把模型做宽（增加专家数量）；同时通过参数共享和 recurrence 来控制参数总量并提升效率。**

它提出了名为 WideNet 的结构，想解决的主要问题是，如何**在压缩模型参数量的情况下取得更好的效果**。比如Albert通过参数共享机制降低了BERT的参数量，像tiny-bert之类的则是减少了Transformer的层数，但他们的性能都有了显著的下降。这篇文章提出，**首先通过层之间的参数共享，来压缩模型大小，然后我们使用MoE的设计，扩大模型容量**（但是模型在feed forward的时候计算量并没怎么提升），这样就可以达到"既要模型参数少，还要模型效果好"的效果。示意图如下：

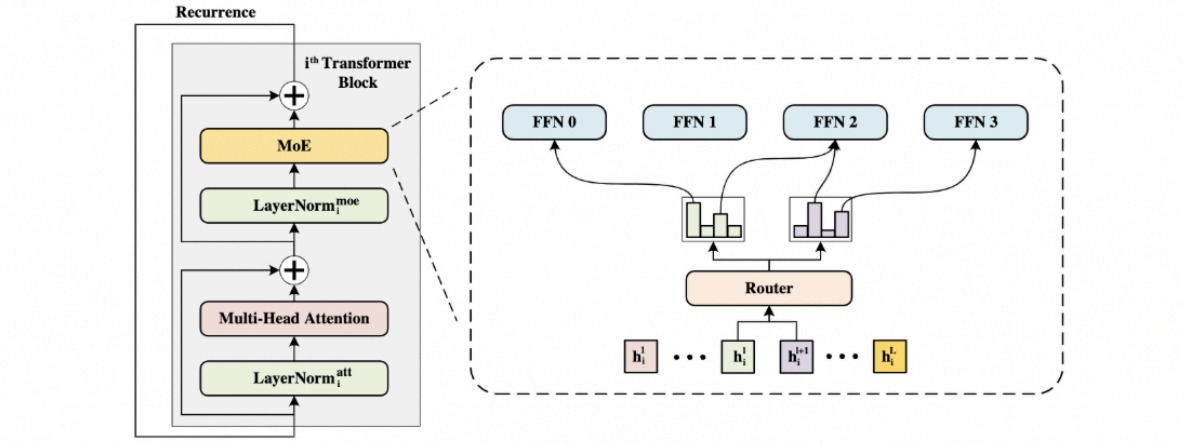


Figure 1: The overall architecture of the proposed WideNet. Compared with vanilla transformer, we replace FFN layer by MoE layer and share the trainable parameters except the normalization layers.

咋一看，似乎跟前面几个文章一模一样，但这里有一个重要区别：**使用了recurrence机制**，即层之间的参数共享（MoE layer也共享）。另外，为了增加学习的多样性，**normalization layer 并不共享**。因为多次输入到同一个 LayerNorm 会导致数值分布不稳定，并且相邻"虚拟层"之间的表示差异会变小，影响学习能力。具体实现时，这里使用总共4个experts，每次选择Top2。

- 所有层使用同一个 gate network 来选择 expert
- 所有层使用同一组 expert（4 个）
- 输入依次多次经过这组 MoE，模拟"多层"的效果

这样做的结果也挺不错：

Model	Parameters	ImageNet-1K
ViT-B	87M	78.6
ViT-L	305M	77.5
ViT-MoE-B	128M	77.9
ViT-MoE-L	406M	77.4
WideNet-B	29M	77.5
WideNet-L	40M	79.5
WideNet-H	63M	**80.1**

Model	#para	SQuAD1.1	SQuAD2.0	MNLI	SST-2	Avg
ALBERT	12M	89.3/82.3	80.0/77.1	81.5	90.3	84.0
BERT	89M	89.9/82.8	80.3/77.3	83.2	91.5	85.0
WideNet 4 experts	26M	89.6/82.7	80.6/77.4	82.6	91.1	84.7
WideNet 8 experts	45M	90.0/82.7	80.6/77.7	83.3	91.9	85.2
WideNet 16 experts	83M	**90.9/83.8**	**81.0/77.9**	**84.1**	**92.2**	**85.8**

完全保留你原始文字内容 的笔记整理如下，仅将符号用 $ $ 表示、公式用 $$ $$ 表示：

3.2 MoEBERT: from BERT to Mixture-of-Experts via Importance-Guided Adaptation, NAACL'22

期刊/会议：NAACL'22

论文链接：https://readpaper.com/paper/4614341372211634177

这一篇文章，则是结合了 MoE 和 knowledge distillation，在提升 inference 速度的情况下，还能提高效果。主要想解决传统的 distillation 方法掉点的问题。具体做法是把一个**预训练好的模型（比如 BERT）**的 FFN 层分解成多个 experts，这样在计算的时候速度可以大幅提高（相当于只激活原始 FFN 网络的一部分）。然后再通过模型蒸馏把原始模型的知识蒸馏到 MoE 版本的模型中。

注意这个文章其实跟上面介绍的 WideNet 类似，也是为了减少参数量。但有一个区别在于，WideNet 是自己从头开始 pre-train 的，但是本文的 MoEBERT 则是想尽可能地把已经 pre-train 好的模型迁移过来，通过 distillation 的方式在 downstream task 上直接学。

因此，如果按照传统的方式让模型自由的去学习不同的 experts，效果可能不好，因为你没有大量的数据来预训练。所以这里涉及到一个关键步骤——**Importance-Guided Adaptation**：在把 Transformer 中的 FFN layer 改造成 MoE layer 时，我们先去计算FFN layer各个神经元（neuron） 的 importance，计算公式如下：

$$I_j = \sum_{(x,y)\in D} \left| (w_j^1)^\top \nabla_{w_j^1} L(x,y) + (w_j^2)^\top \nabla_{w_j^2} L(x,y) \right|$$

- w_j^1 和 w_j^2 分别表示某个神经元在 FFN 层中第一层和第二层中的权重向量；
- $\nabla_{w_j^1} L(x,y)$ 和 $\nabla_{w_j^2} L(x,y)$ 是 loss 对应于这些权重的梯度；
- 整个表达式衡量的是：该神经元的权重在当前任务数据分布下对 loss 的影响程度（即其"重要性"）；

因此：

> 这个 importance score 越高，说明该神经元对模型最终 loss 的影响越大，也就越重要！

这个指标也被广泛应用于模型剪枝（model pruning）中，用于判断哪些神经元可以安全地被移除。而在 MoEBERT 中，它被用来指导如何将原始 FFN 层分解为多个 experts，保留最重要的部分以保持模型性能。

然后，在把 FFN 分解的时候，我们取最重要的一部分 neurons 在每个 expert 中共享，剩下的部分平均分配到每个 expert。由于共享机制的存在，一定会多出一些 neurons，这部分就直接丢弃。（注意，这里我们并没有增加模型的参数量，而只是把一个全连接的 FFN 层，分解成多个 sub-networks，加起来的参数量实际上是一样的）

这个示意图很形象：

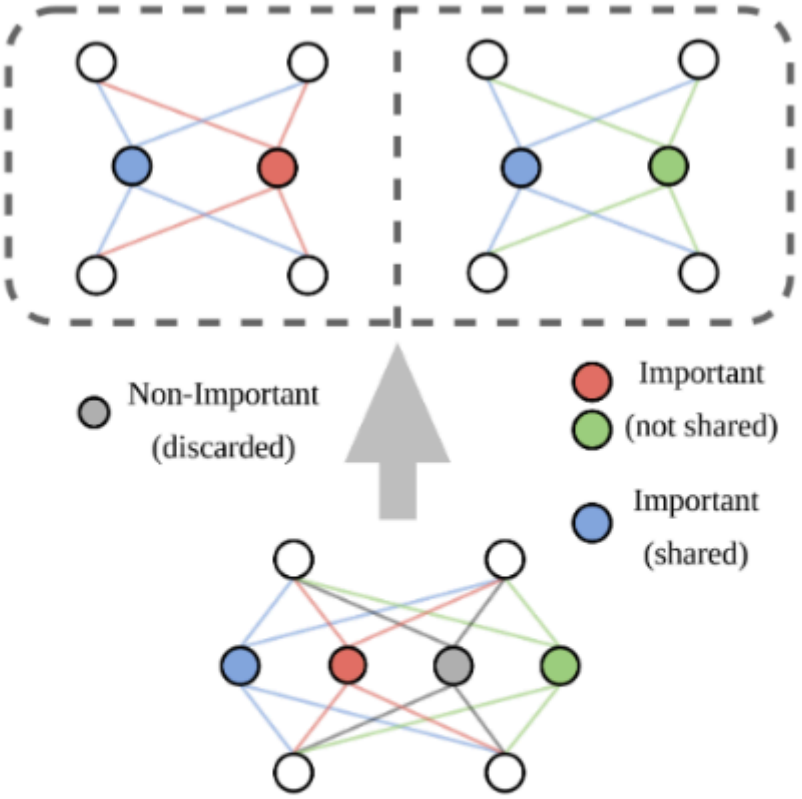


Figure 1: Adapting a two-layer FFN into two experts. The blue neuron is the most important one, and is shared between the two experts. The red and green neurons are the second and third important ones, and are assigned to expert one and two, respectively.

MoEBERT 没有使用 gating network 来动态选择 expert，而是在训练前就为每个 token 固定分配一个 expert（通过 hash 函数实现），从而简化了 routing 机制。

在distillation部分，这里使用的逐层的distillation MSE loss，以及最后预测概率的 KL loss，二者加起来就是distillation 所使用的 loss。然后，再和原本自己的 CE loss 加起来，就是总体模型训练的loss。
具体如下：

1.Layer-wise Distillation（逐层蒸馏）：对每一层输出的 hidden states 进行监督，使用 MSE loss 衡量学生和教师之间的差异：

$$L_{\text{mse}} = \frac{1}{T} \sum_{t=1}^{T} \| h_t^{\text{student}} - h_t^{\text{teacher}} \|^2$$

其中：

- h_t 是第 t 层的 hidden state
- T 是总的层数

2.Output-level Distillation（输出层蒸馏）：在最终输出 logits 上进行 KL 散度损失（soft label 蒸馏）：

$$L_{\text{kl}} = D_{\text{KL}}(p_{\text{teacher}} \parallel p_{\text{student}})$$

3.总体 Distillation Loss：

$$L_{\text{distill}} = \alpha \cdot L_{\text{mse}} + \beta \cdot L_{\text{kl}}$$

通常会设置 $\alpha = 1, \beta = 1$，也可以调参。

4.最终训练 Loss：除了蒸馏 loss，MoEBERT 还保留了下游任务本身的分类 loss（CrossEntropy）：

$$L_{\text{total}} = L_{\text{CE}} + \gamma \cdot L_{\text{distill}}$$

其中：

- L_{CE}：下游任务的真实标签交叉熵损失
- γ：控制蒸馏 loss 的权重，默认设为 1

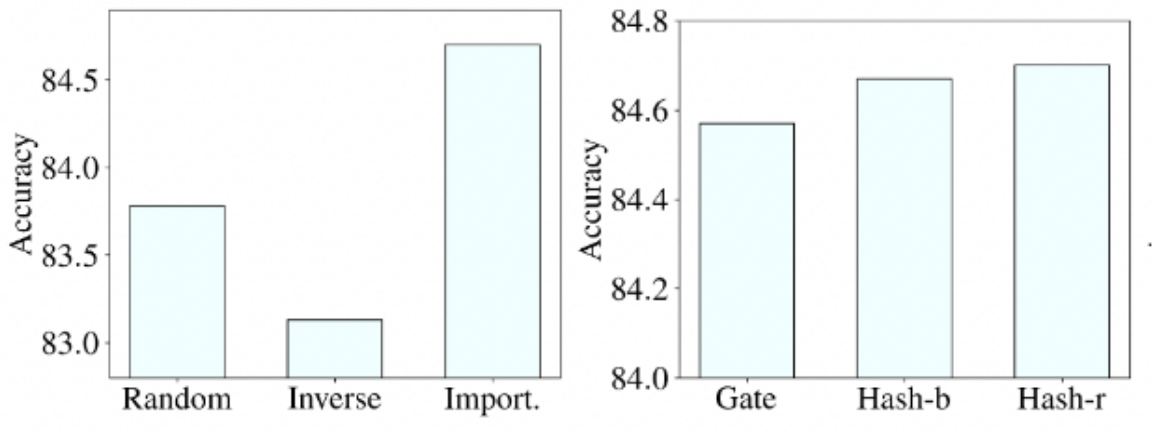
这里是直接在downstream dataset上面进行训练，属于 task-specific distillation（任务特定蒸馏）。

	RTE Acc	CoLA Mcc	MRPC F1/Acc	SST-2 Acc	QNLI Acc	QQP F1/Acc	MNLI m/mm
BERT-base	63.5	54.7	89.0/84.1	92.9	91.1	88.3/90.9	84.5/84.4
Task-agnostic							
DistilBERT	59.9	51.3	87.5/-	92.7	89.2	-/88.5	82.2/-
TinyBERT (w/o aug)	72.2	42.8	88.4/-	91.6	90.5	-/90.6	83.5/-
MiniLMv1	71.5	49.2	88.4/-	92.0	91.0	-/91.0	84.0/-
MiniLMv2	72.1	52.5	88.9/-	92.4	90.8	-/91.1	84.2/-
CoDIR (pre+fine)	67.1	53.7	89.6/-	**93.6**	90.1	-/89.1	83.5/82.7
Task-specific							
PKD	65.5	24.8	86.4/-	92.0	89.0	-/88.9	81.5/81.0
BERT-of-Theseus	68.2	51.1	89.0/-	91.5	89.5	-/89.6	82.3/-
CoDIR (fine)	65.6	53.6	89.4/-	**93.6**	90.4	-/89.1	83.6/82.8
Ours (task-specific)							
MoEBERT	**74.0**	**55.4**	**92.6/89.5**	93.0	**91.3**	**88.4/91.4**	**84.5/84.8**

Table 1: Experimental results on the GLUE development set. The best results are shown in **bold**. All the models are trained without data augmentation. All the models have $66M$ parameters, except BERT-base ($110M$ parameters). We report mean over three runs. Model references: BERT (Devlin et al., 2019), DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2020), MiniLMv1 (Wang et al., 2020), MiniLMv2 (Wang et al., 2021), CoDIR (Sun et al., 2020a), PKD (Sun et al., 2019), BERT-of-Theseus (Xu et al., 2020).

实验的结果也验证了 MoEBERT可以在同样参数量（effective parameters，MoE layer中只考虑被激活的experts）的情况下超越其他 distillation baselines，即使这些 baseline 模型使用了预训练。

值得注意的时，这里的baselines中，task-agnostic（**与任务无关**）的方法都使用了预训练，而task-specific都没有预训练。总体上看，使用了预训练的模型，效果都会更好一些，但是MoEBERT打破了这个规律，在只使用task dataset的情况下，取得了SOTA的结果。



(a) Adaptation methods. (b) Routing methods in MoE.

图a验证了前面提到的 Importance-Guided Adaptation 的有效性；图b则是验证了通过hash function的方式，而不是 trainable gating的方式来进行routing 的有效性。

4. ST-MOE

之前讨论的负载均衡损失可能会导致稳定性问题。我们可以使用许多方法来稳定稀疏模型的训练，但这可能会牺牲模型质量。例如，引入 dropout 可以提高稳定性，但会导致模型质量下降。

4.1 用 Router z-loss 稳定模型训练

在论文 ST-MOE: Designing Stable and Transferable Sparse Expert Models 中，作者提出了一种新的辅助损失函数，称为 **Router z-loss**，用于提高稀疏模型的训练稳定性，同时保持或稍微提高模型质量。这个损失函数是针对稀疏专家模型中的路由器（router）部分设计的，路由器负责将输入的 token 路由到最合适的专家（expert）层。

在 MoE 模型中，每个输入 token 可能被路由到多个专家，但通常只有一个专家层会被激活。为了确保路由器能够稳定地工作并产生高质量的输出，**作者引入了 Router z-loss。这个损失函数的目标是鼓励路由器产生较小的 logits 值，因为较大的 logits 值在 softmax 激活函数中会导致较大的梯度，这可能会引起训练不稳定。**

Router z-loss 是一种鼓励路由器输出稳定 logits 的辅助损失函数，它通过对 logits 的指数和取 log 后平方，惩罚那些过大或分布不均的 logit 值，从而提升 MoE 模型的训练稳定性与泛化能力。Router z-loss 的定义如下：

$$L_z(x) = \frac{1}{B}\sum_{i=1}^{B}\left(\log\sum_{j=1}^{N} e^{x_j^{(i)}}\right)^2$$

其中，B 是 batch 中的 token 数量，N 是专家的数量，$x \in \mathbb{R}^{B\times N}$ 是路由器的 logits。这个损失函数通过惩罚较大的 logits 值来工作，因为这些值在 softmax 函数中会导致较大的梯度。通过这种方式，Router z-loss 有助于减少训练过程中的不稳定性，并可能提高模型的泛化能力。

4.2 专家如何学习?

ST-MoE 的研究者们发现，Encoder 中不同的专家倾向于专注于特定类型的 token 或浅层概念。例如，某些专家可能专门处理标点符号，而其他专家则专注于专有名词等。与此相反，Decoder 中的专家通常具有较低的专业化程度。此外，研究者们还对这一模型进行了多语言训练。尽管人们可能会预期每个专家处理一种特定语言，但实际上并非如此。由于 token 路由和负载均衡的机制，没有任何专家被特定配置以专门处理某一特定语言。

4.3 专家的数量对预训练有何影响?

增加更多专家可以在一定程度上提升模型处理样本的效率并加快运算速度。然而，这种优势随着专家数量的增加而逐渐减弱，尤其是在专家数量达到 256 或 512 后，边际效益明显下降。与此同时，更多的专家意味着在推理过程中需要加载更多参数，从而导致显存占用显著增加。

值得注意的是，Switch Transformer 的研究表明，MoE 结构在大规模模型中展现的优势同样适用于小规模模型。即使每层仅包含 2、4 或 8 个专家，MoE 模型也能在下游任务中表现出明显的性能提升。例如，在 GLUE 等自然语言理解任务中，即便每层只使用 2~8 个 expert，MoE 模型的表现依然显著优于传统的 dense 模型。

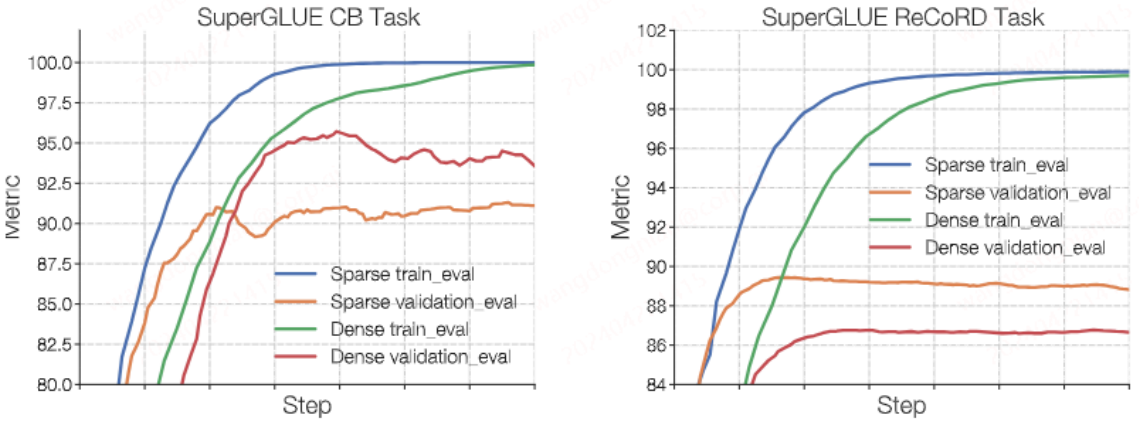
由此可见，MoE 并不是“仅适用于超大模型”的技术，它同样可以有效地应用于中等甚至小型模型，带来推理效率和模型容量的双重提升。

4.4 Fine-Tuning MoE 模型

稠密模型和稀疏模型在过拟合的动态表现上存在显著差异。**稀疏模型更易于出现过拟合现象**，因此在处理这些模型时，尝试更强的内部正则化措施是有益的，比如**使用更高比例的 dropout**。例如，可以为稠密层设定一个较低的 dropout 率，而为稀疏层设置一个更高的 dropout 率，以此来优化模型性能。

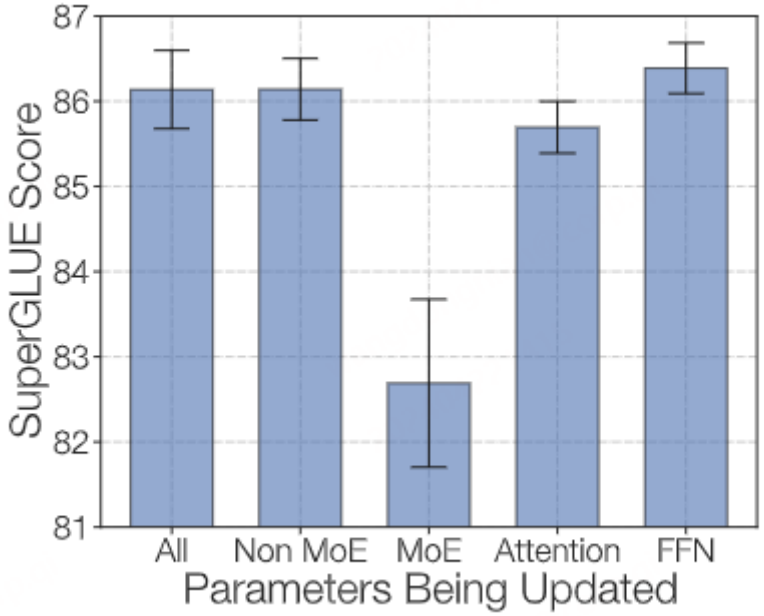
在 Fine-Tuning 过程中是否使用辅助损失是一个需要决策的问题。ST-MoE 的作者尝试关闭辅助损失，发现即使高达 11% 的 token 被丢弃，模型的质量也没有显著受到影响。token 丢弃可能是一种正则化形式，有助于防止过拟合。

实验观察到，在相同的预训练 PPL 下，稀疏模型在下游任务中的表现不如对应的稠密模型，特别是在理解任务 (如 SuperGLUE) 上。另一方面，对于知识密集型任务 (如 TriviaQA)，稀疏模型的表现异常出色。作者还观察到，在Fine-Tuning 过程中，较少的专家的数量有助于改善性能。另一个关于泛化问题确认的发现是，模型在小型任务上表现较差，但在大型任务上表现良好。



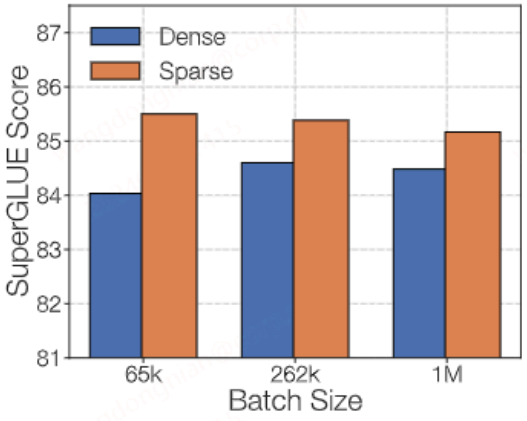
> **Tip**
>
> 在小任务 (左图) 中，我们可以看到明显的过拟合，因为稀疏模型在验证集中的表现要差得多。在较大的任务 (右图) 中，MoE 则表现良好。

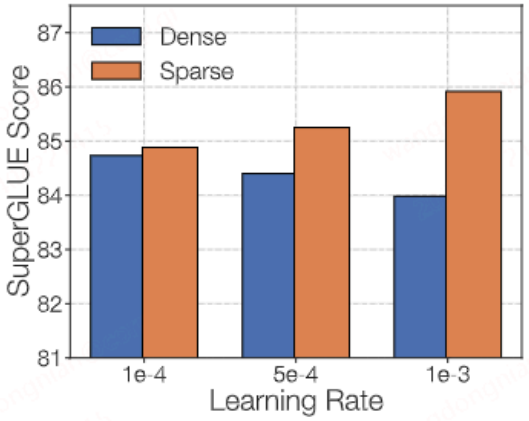
一种可行的 Fine-Tuning 策略是尝试冻结所有非专家层的权重。实践中，这会导致性能大幅下降，可以尝试相反的方法：冻结 MoE 层中的参数（包括 gating network 和 experts），仅更新其他层（如 attention、layer norm、embedding 等）。实验结果显示，这种方法几乎与更新所有参数的效果相当。这种做法可以加速 Fine-Tuning 过程，并降低显存需求。



> **Tip**
>
> 通过仅冻结 MoE 层，我们可以在保持模型效果的同时加快训练速度

在 Fine-Tuning MoE 时还需要考虑的一个问题是，它们有需要特殊设置的超参数，例如，**稀疏模型往往更适合使用较小的 batch size 和较高的学习率**，这样可以获得更好的训练效果。





> **Tip**
>
> 提高学习率和降低batch size可以提升稀疏模型微调效果

5.结语

以上总结了一下笔者在阅读 MoE 相关文献时印象较深的几篇文章，上述所阅读的文献主要与NLP相关的，其实 MoE 在各个领域中的应用已经十分广泛。比如Google提出的多模态MoE模型——LIMoE：

另外，跟 MoE 的理念相关的还有很多有趣的工作，比如：

Diverse Ensemble Evolution: Curriculum Data-Model Marriage, NeurIPS'18

Diversity and Depth in Per-Example Routing Models, ICLR'21

MoE 的思想，其实十分符合 Google 提出的 Pathways 愿景，也更加符合通用人工智能的设计理念。虽然目前 MoE 的工作，多数都是开发“超级模型”，但是上面列举的一些工作也表明 MoE 的用途还有很多，可以启发很多方向上方法的改进。