

异常庞大的神经网络：稀疏门控的专家混合层

Noam Shazeer¹, Azalia Mirhoseini^{*†1}, Krzysztof Maziarz^{*2}, Andy Davis¹, Quoc Le¹, Geoffrey Hinton¹ and Jeff Dean¹

¹Google Brain, {noam,azalia,andydavis,qvl,geoffhinton.jeff}@google.com

²Jagiellonian University, Cracow, krzysztof.maziarz@student.uj.edu.pl

ABSTRACT

神经网络吸收信息的能力受其参数数量的限制。条件计算，其中网络的某些部分在每个示例的基础上都是活跃的，在理论上已经被提出作为一种在不按比例增加计算量的情况下显着增加模型容量的方法。然而，在实践中，存在重大的算法和性能挑战。在这项工作中，我们解决了这些挑战并最终实现了条件计算的承诺，在模型容量方面实现了 1000 倍以上的改进，而在现代 GPU 集群上的计算效率只有很小的损失。我们引入了一个稀疏门控混合专家层 (MoE)，由多达数千个前馈子网络组成。可训练的门控网络确定这些专家的稀疏组合以用于每个示例。我们将 MoE 应用于语言建模和机器翻译的任务，其中模型容量对于吸收训练语料库中可用的大量知识至关重要。我们展示了模型架构，其中在堆叠的 LSTM 层之间卷积应用具有多达 1370 亿个参数的 MoE。在大型语言建模和机器翻译基准测试中，这些模型以更低的计算成本取得了比最先进的更好的结果。

1 引言及相关工作

1.1 条件计算

利用训练数据和模型大小的规模一直是深度学习成功的核心。当数据集足够大时，增加神经网络的容量（参数数量）可以提供更好的预测精度。这已在文本 (Sutskever 等人, 2014 年; Bahdanau 等人, 2014 年; Jozefowicz 等人, 2016 年; Wu 等人, 2016 年)、图像 (Krizhevsky 等人, 2012 年; Le 等人) 等领域得到体现 al., 2012) 和音频 (Hinton et al., 2012; Amodei et al., 2015)。对于典型的深度学习模型，每个示例都会激活整个模型，随着模型大小和训练示例数量的增加，这会导致训练成本大致呈二次增长。不幸的是，计算能力和分布式计算的进步不能满足这种需求。

已经提出各种形式的条件计算作为增加模型容量而不按比例增加计算成本的方法 (Davis & Arel, 2013; Bengio et al., 2013; Eigen et al., 2013; Ludovic Denoyer, 2014; Cho & Bengio, 2014 年; Bengio 等人, 2015 年; Almahairi 等人, 2015 年)。在这些方案中，网络的大部分在每个示例的基础上处于活动状态或非活动状态。门控决策可以是二元的或稀疏的和连续的、随机的或确定性的。提出了各种形式的强化学习和反向传播来训练门控决策。

* 同等主要贡献者 † 作为 Google Brain Residency 计划
(g.co/brainresidency) 成员所做的工作

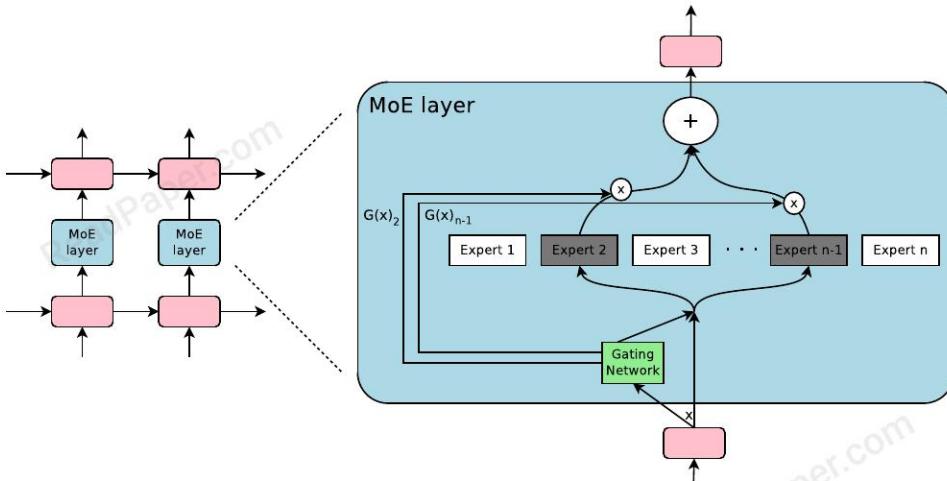


图 1：嵌入循环语言模型中的混合专家（MoE）层。在这种情况下，稀疏门函数选择两个专家来执行计算。它们的输出由门控网络的输出调制。

虽然这些想法在理论上很有前途，但迄今为止还没有任何工作证明模型容量、训练时间或模型质量有显著改善。我们将此归咎于以下挑战的组合：

- 现代计算设备，尤其是GPU，在算术上比在分支上快得多。上面的大部分工作都认识到这一点，并建议在每个门控决策中打开/关闭网络的大块。
- 大批量对性能至关重要，因为它们分摊了参数传输和更新的成本。条件计算减少了网络有条件活动块的批量大小。
- 网络带宽可能成为瓶颈。GPU集群的计算能力可能是设备间网络总带宽的数千倍。为了提高计算效率，算法的相对计算需求与网络需求必须超过此比率。嵌入层，可以看作是一种条件计算的形式，受到这个问题的阻碍。由于嵌入通常需要通过网络发送，因此（例如，参数）交互的数量受网络带宽而不是计算能力的限制。
- 根据方案的不同，损失项可能是实现每个块和/或每个示例所需的稀疏程度所必需的。本吉奥等。（2015）使用了三个这样的术语。这些问题会影响模型质量和负载平衡。
- 模型容量对于非常大的数据集来说最为关键。现有的条件计算文献涉及相对较小的图像识别数据集，最多包含 600,000 张图像。很难想象这些图像的标签提供了足够的信号来充分训练具有数百万个参数的模型，更不用说数十亿个参数了。

在这项工作中，我们首次解决了上述所有挑战，并最终实现了条件计算的承诺。我们在模型容量上获得了超过 1000 倍的改进，而计算效率只有很小的损失，并且显着提高了公共语言建模和翻译数据集的最先进结果。

1.2 OUR APPROACH: THE SPARSELY-GATED MIXTURE-OF-EXPERTS LAYER

我们的条件计算方法是引入一种新型的通用神经网络组件：稀疏门控混合专家层（MoE）。MoE 由许多专家组成，每个专家都是一个简单的前馈神经网络，以及一个可训练的门控网络，该门控网络选择专家的稀疏组合来处理每个输入（见图 1）。网络的所有部分都通过反向传播联合训练。

虽然介绍的技术是通用的，但在本文中，我们专注于语言建模和机器翻译任务，众所周知，这些任务受益于非常大的模型。特别是，我们在堆叠的 LSTM 层之间卷积应用 MoE (Hochreiter & Schmidhuber, 1997)，如图 1 所示。MoE 为文本中的每个位置调用一次，在每个位置选择可能不同的专家组合。不同的专家往往根据句法和语义变得高度专业化（参见附录 E 表 9）。在语言建模和机器翻译基准测试中，我们以一小部分计算成本改进了最佳发表结果。

1.3 专家组合的相关工作

自二十多年前推出以来 (Jacobs 等人, 1991 年; Jordan 和 Jacobs, 1994 年)，专家混合方法一直是许多研究的主题。已经提出了不同类型的专家架构，例如 SVM (Collobert 等人, 2002 年)、高斯过程 (Tresp, 2001 年; Theis 和 Bethge, 2015 年; Deisenroth 和 Ng, 2015 年)、狄利克雷过程 (Shahbaba 和 Neal, 2009 年)、和深度网络。其他工作侧重于不同的专家配置，例如层次结构 (Yao 等人, 2009 年)、无限数量的专家 (Rasmussen & Ghahramani, 2002 年) 以及按顺序添加专家 (Aljundi 等人, 2016 年)。Garmash & Monz (2016) 建议采用专家混合格式的集成模型用于机器翻译。门控网络是在预训练的集成 NMT 模型上训练的。

上述工作涉及顶级专家组合。专家的混合就是整个模型。本征等。 (2013) 介绍了使用具有自己的门控网络的多个 MoE 作为深度模型的一部分的想法。直觉上后一种方法更强大，因为复杂的问题可能包含许多子问题，每个子问题都需要不同的专家。他们还在结论中提到引入稀疏性的潜力，将 MoE 转变为计算工具。

我们的工作建立在将 MoE 用作通用神经网络组件的基础上。而 Eigen 等人。 (2013) 使用两个堆叠的 MoE 允许两组门控决策，我们对 MoE 的卷积应用允许在文本中的每个位置进行不同的门控决策。我们还实现了稀疏门控，并展示了它作为一种实用方法来大规模增加模型容量的用途。

2 专家混合层的结构

混合专家 (MoE) 层由一组 n 个“专家网络” E_1, \dots, E_n 和一个输出为稀疏 n 维向量的“门控网络” G 组成。图 1 显示了 MoE 模块的概览。专家本身就是神经网络，每个都有自己的参数。虽然原则上我们只要求专家接受相同大小的输入并产生相同大小的输出，但在本文的初步调查中，我们将自己限制在模型是具有相同架构的前馈网络的情况下，但单独的参数。

让我们用 $G(x)$ 和 $E_i(x)$ 表示给定输入 x 的门控网络的输出和第 i 个专家网络的输出。 MoE 模块的输出 y 可以写成如下：

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

我们基于 $G(x)$ 输出的稀疏性来节省计算。只要 $G(x)_i = 0$ ，我们就不需要计算 $E_i(x)$ 。在我们的实验中，我们有多达数千名专家，但只需要为每个示例评估少数专家。如果专家数量非常多，我们可以通过使用两级分层 MoE 来减少分支因子。在分层 MoE 中，初级门控网络选择“专家”的稀疏加权组合，每个“专家”本身都是具有自己的门控网络的二级专家混合体。下面我们重点关注普通 MoE。我们提供更多细节关于附录 B 中的分层 MoE。

我们的实现与其他条件计算模型有关。其专家是简单权重矩阵的 MoE 类似于 (Cho & Bengio, 2014) 中提出的参数化权重矩阵。其专家有一个隐藏层的 MoE 类似于 (Bengio et al., 2015) 中描述的逐块丢失，其中丢失层夹在完全激活层之间。

2.1 门控网络

Softmax 门控：非稀疏门控函数的一种简单选择 (Jordan & Jacobs, 1994) 是将输入乘以可训练的权重矩阵 W_g , 然后应用 Softmax 函数。

$$G_\sigma(x) = \text{Softmax}(x \cdot W_g) \quad (2)$$

嘈杂的 Top-K 门控：我们向 Softmax 门控网络添加了两个组件：稀疏性和噪声。在采用 softmax 函数之前，我们添加可调高斯噪声，然后仅保留前 k 个值，将其余值设置为 $-\infty$ (这会导致相应的门值等于 0)。如上所述，稀疏性用于节省计算。虽然这种形式的稀疏性在门函数的输出中造成了一些理论上可怕的不连续性，但我们还没有观察到这在实践中是一个问题。噪声项有助于负载平衡，这将在附录 A 中讨论。每个组件的噪声量由第二个可训练权重矩阵 W_{noise} 控制。

$$G(x) = \text{Softmax}(\text{KeepTopK}(H(x), k)) \quad (3)$$

$$H(x)_i = (x \cdot W_g)_i + \text{StandardNormal()} \cdot \text{Softplus}((x \cdot W_{noise})_i) \quad (4)$$

$$\text{KeepTopK}(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ -\infty & \text{otherwise.} \end{cases} \quad (5)$$

训练门控网络我们通过简单的反向传播以及模型的其余部分来训练门控网络。如果我们选择 $k > 1$, 则前 k 个专家的门值相对于门控网络的权重具有非零导数。这种偶尔敏感的行为在 (Bengio et al., 2013) 中关于噪声整流器进行了描述。梯度也通过门控网络反向传播到它的输入。我们的方法不同于 (Bengio et al., 2015), 后者使用布尔门和 REINFORCE 风格的方法来训练门控网络。

3 应对性能挑战

3.1 收缩批量问题

在现代 CPU 和 GPU 上，大批量大小对于计算效率是必要的，以便分摊参数加载和更新的开销。如果门控网络为每个示例从 n 个专家中选择 k 个，那么对于一批 b 个示例，每个专家收到一个小得多的批次，大约 $\frac{b}{k}$ 例子。随着专家数量的增加，这会导致简单的 MoE 实现变得非常低效。解决这个缩小批量问题的方法是让原始批量大小尽可能大。然而，批量大小往往受到存储向前和向后传递之间的激活所需的内存的限制。我们提出了以下增加批量大小的技术：

混合数据并行和模型并行：在传统的分布式训练设置中，不同设备上的多个模型副本异步处理不同批次的数据，并通过一组参数服务器同步参数。在我们的技术中，这些不同的批次同步运行，以便它们可以组合用于 MoE 层。我们根据传统的数据并行方案分配模型和门控网络的标准层，但每个专家只保留一份共享副本。MoE 层中的每个专家都会收到一个组合批次，其中包含来自所有数据并行输入批次的相关示例。同一组设备用作数据并行副本（用于标准层和门控网络）和模型并行分片（每个分片托管专家的子集）。如果模型分布在 d 个设备上，并且每个设备处理一个大小为 b 的批次，则每个专家收到大约

$\frac{n}{d}$ 个例子。因此，我们实现了专家批量大小的 d 改进。

在分层 MoE (B 部分) 的情况下，主要门控网络采用数据并行性，次要 MoE 使用模型并行性。每个辅助 MoE 驻留在一个设备上。

这种技术允许我们通过按比例增加训练集群中的设备数量来增加专家的数量（以及参数的数量）。总批量大小增加，保持每个专家的批量大小不变。每个设备的内存和带宽要求也保持不变，步骤时间也是如此，处理与模型中参数数量相等的训练示例所需的时间量也是如此。我们的目标是在万亿词语料库上训练万亿参数模型。在撰写本文时，我们还没有将系统扩展到这种程度，但应该可以通过添加更多硬件来实现。

利用卷积：在我们的语言模型中，我们将相同的 MoE 应用于前一层的每个时间步长。如果我们等待上一层完成，我们可以将 MoE 作为一个大批量一起应用于所有时间步长。这样做会增加 MoE 层的输入批次的大小，乘以展开时间步长的数量。

增加循环 MoE 的批量大小：我们怀疑更强大的模型可能涉及循环应用 MoE。例如，LSTM 或其他 RNN 的权重矩阵可以用 MoE 代替。遗憾的是，此类模型打破了上一段中的卷积技巧，因为一个时间步的 MoE 输入取决于前一时间步的 MoE 输出。Gruslys 等人。(2016) 描述了一种以重新计算前向激活为代价大幅减少展开 RNN 中存储的激活数量的技术。这将允许批量大小的大幅增加。

3.2 网络带宽

分布式计算中另一个主要的性能问题是网络带宽。由于专家是固定的（见上文）并且门控参数的数量很少，因此大部分通信涉及通过网络发送专家的输入和输出。为了保持计算效率，专家的计算与其输入和输出大小的比率必须超过计算设备的计算能力与网络容量的比率。对于 GPU，这可能是千分之一。在我们的实验中，我们使用一个隐藏层包含数千个 RELU 激活单元的专家。由于 expert 中的权重矩阵大小为 $\text{input_size} \times \text{hidden_size}$ 和 $\text{hidden_size} \times \text{output_size}$ ，因此计算输入和输出的比率等于隐藏层的大小。方便地，我们可以简单地通过使用更大的隐藏层或更多隐藏层来提高计算效率。

4 平衡专家的使用

我们已经观察到，门控网络倾向于收敛到一种状态，在这种状态下，它总是为相同的少数专家产生较大的权重。这种不平衡是自我强化的，因为受青睐的专家训练得更快，因此被门控网络选择得更多。本征等。(2013) 描述了相同的现象，并在训练开始时使用硬约束来避免这种局部最小值。本吉奥等。(2015) 包括对每个门的批处理平均值的软约束。¹

我们采用软约束方法。我们将专家相对于一批训练示例的重要性定义为该专家的门值的批次总和。我们定义了一个额外的损失 $L_{\text{importance}}$ ，它被添加到模型的整体损失函数中。该损失等于重要性值集合的变异系数的平方乘以手动调整的比例因子 $w_{\text{importance}}$ 。这种额外的损失鼓励所有专家具有同等重要性。

$$\text{Importance}(X) = \sum_{x \in X} G(x) \quad (6)$$

$$L_{\text{importance}}(X) = w_{\text{importance}} \cdot \text{CV}(\text{Importance}(X))^2 \quad (7)$$

¹ 本吉奥等人。(2015) 还包括两个额外的损失。一个控制每个示例的稀疏性，我们不需要它，因为它由 k 的固定值强制执行。第三个损失鼓励门值的多样性。在我们的实验中，我们发现门值随着专家的专业化（良性循环）自然地多样化，我们不需要强制门值的多样性。

虽然这个损失函数可以确保同等重要性，但专家们可能仍然会收到非常不同数量的例子。例如，一位专家可能会收到一些权重较大的示例，而另一位专家可能会收到许多权重较小的示例。这可能会导致分布式硬件出现内存和性能问题。为了解决这个问题，我们引入了第二个损失函数 L_{load} ，它可以确保负载均衡。附录 A 包含此函数的定义以及实验结果。

5个实验

5.1 十亿字语言建模基准

数据集：该数据集由 (Chelba et al., 2013) 引入，由新闻文章中经过打乱的独特句子组成，总计约 8.29 亿个单词，词汇量为 793,471 个单词。

以前的最新技术：以前发表的最佳结果 (Jozefowicz 等人, 2016 年) 使用由一个或多个堆叠的长短期记忆 (LSTM) 层组成的模型 (Hochreiter & Schmidhuber, 1997 年; Gers 等人。, 2000). 这些模型的 LSTM 层中的参数数量从 200 万到 1.51 亿不等。质量随着参数数量的增加而大大增加，计算成本也是如此。这些模型的结果构成了图 2 右侧的顶行。

MoE 模型：我们的模型由两个堆叠的 LSTM 层组成，它们之间有一个 MoE 层（见图 1）。我们改变层的大小和专家的数量。有关模型架构、训练方案、其他基线和结果的完整详细信息，请参阅附录 C。

低计算量，可变容量：为了研究增加容量的影响，我们训练了一系列 MoE 模型，所有这些模型的计算成本大致相同：前向传递中每个时间步的每个训练示例约 800 万次乘法和加法，不包括 softmax 层。我们称此指标为 (ops/timestep)。我们使用包含 4、32 和 256 位专家的平面 MoE 训练模型，并使用包含 256、1024 和 4096 位专家的分层 MoE 训练模型。每个专家有大约 100 万个参数。对于所有 MoE 层，每个输入有 4 个专家处于活动状态。

这些模型的结果如图 2-左所示。具有 4 名始终活跃的专家的模型（不出所料）与计算匹配的基线模型的表现相似，而最大的模型（4096 名专家）在测试集上的困惑度降低了 24%，令人印象深刻。

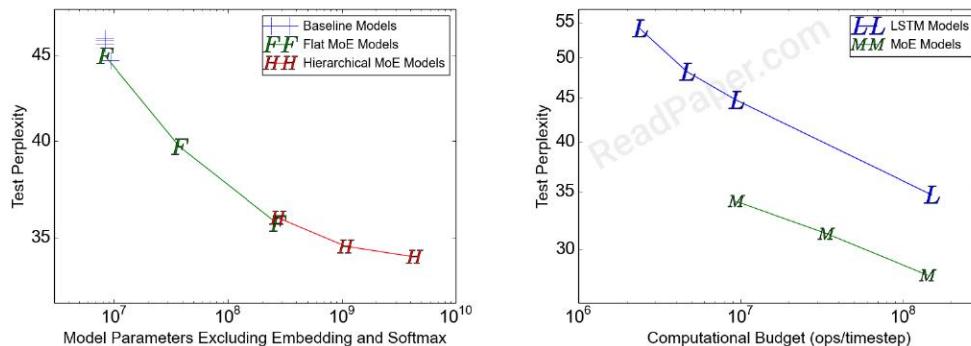


图 2：10 亿字语言建模基准的模型比较。在左侧，我们将测试复杂度绘制为模型容量的函数，这些模型具有类似的计算预算（每个时间步约 800 万次操作）。在右侧，我们将测试困惑度绘制为计算预算的函数。顶行代表来自 (Jozefowicz et al., 2016) 的 LSTM 模型。底行代表具有不同计算预算的 40 亿个参数 MoE 模型。

可变计算，高容量：除了上一节中最大的模型外，我们还训练了两个具有类似高容量（40 亿个参数）但计算预算更高的 MoE 模型。这些模型有更大的 LSTM，数量更少但更大且更专业。细节

表 1：具有不同计算预算的高容量 MoE 增强模型与先前发布的最佳结果的总结 (Jozefowicz 等人, 2016 年)。详见附录 C。

	Test Perplexity 10 epochs	Test Perplexity 100 epochs	#Parameters excluding embedding and softmax layers	ops/timestep	Training Time 10 epochs	TFLOPS /GPU
Best Published Results	34.7	30.6	151 million	151 million	59 hours, 32 k40s	1.09
Low-Budget MoE Model	34.1		4303 million	8.9 million	15 hours, 16 k40s	0.74
Medium-Budget MoE Model	31.3		4313 million	33.8 million	17 hours, 32 k40s	1.22
High-Budget MoE Model	28.0		4371 million	142.7 million	47 hours, 32 k40s	1.56

可以在附录 C.2 中找到。这三个模型的结果构成了图 2-right 的底线。表 1 将这些模型的结果与该数据集上先前发布的最佳结果进行了比较。尽管只需要 6% 的计算量，但即使是这些模型中最快的模型也能打败已发布的最佳结果（在控制训练时期的数量时）。

计算效率：我们使用 TensorFlow (Abadi 等人, 2016 年) 在包含 16-32 个 Tesla K40 GPU 的集群上训练我们的模型。对于我们的每个模型，我们通过将处理一个训练批次所需的浮点运算数量除以观察到的步长时间和集群中的 GPU 数量来确定 TFLOPS/GPU 的计算效率。这里使用的操作计数高于我们在操作/时间步数中报告的操作计数，因为我们包括向后传递，我们包括 softmax 层的基于重要性采样的训练，并且我们将乘法和加法计算为两个独立的操作。对于我们所有的 MoE 模型，专家涉及的浮点运算占总数的 37% 到 46%。

对于我们没有 MoE 的基线模型，观察到的计算效率范围为 1.07-1.29 TFLOPS/GPU。对于我们的低计算量 MoE 模型，计算效率范围为 0.74-0.90 TFLOPS/GPU，但没有充分利用可用并行性的 4 专家模型除外。我们最高计算量的 MoE 模型在 1.56 TFLOPS/GPU 时更高效，这可能是由于更大的矩阵。这些数字代表 NVIDIA 声称的理论最大值 4.29 TFLOPS/GPU 的重要部分。详细结果见附录 C，表 7。

5.2 千亿词谷歌新闻语料库

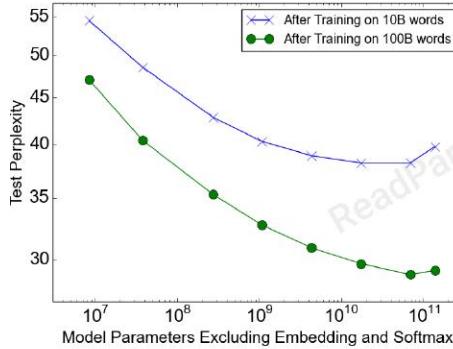


图 3：1000 亿词语料库的语言建模。模型具有相似的计算预算 (800 万次操作/时间步长)。

在 10 亿词的语料库中，随着 MoE 层中的参数数量超过 10 亿，增加额外的容量似乎会产生递减的收益，如图 2-left 所示。我们假设对于更大的训练集，更高的容量会产生显着的质量改进。

我们构建了一个类似的训练集，其中包含来自 Google 内部新闻语料库的随机排列的独特句子，总计约 1000 亿个单词。与上一节类似，我们测试了一系列模型，这些模型的计算成本相似，约为 800 万次操作/时间步长。除了基线 LSTM 模型外，我们还训练了包含 32、256、1024、

4096、16384、65536、131072专家。这对应于MoE层中多达1370亿个参数。有关体系结构、训练和结果的详细信息，请参见附录D。

结果：图3显示了在对100亿个单词（顶线）和1000亿个单词（底线）进行训练后，测试困惑度与容量的函数关系。当对全部1000亿个单词进行训练时，测试困惑度显著提高到65536名专家（680亿个参数），比计算匹配的基线下降39%，但在131072名专家处下降，这可能是稀疏性过多的结果。两条线之间不断扩大的差距表明（不出所料）增加模型容量有助于更大的训练集。

即使在65536位专家（99.994%的层稀疏度）下，模型的计算效率也保持在可观的0.72TFLOPS/GPU。

5.3 机器翻译（单一语言对）

模型架构：我们的模型是(Wu et al., 2016)中描述的GNMT模型的修改版本。为了减少计算量，我们将编码器和解码器中的LSTM层数分别从9层和8层减少到3层和2层。我们在编码器（第2层和第3层之间）和解码器（第1层和第2层之间）中插入了MoE层。每个MoE层包含多达2048个专家，每个专家有大约200万个参数，总共为模型添加了大约80亿个参数。有关模型架构、测试程序和结果的更多详细信息，请参见附录E。

数据集：我们在WMT'14 En→Fr和En→De语料库上对我们的方法进行了基准测试，其训练集分别有36M句对和5M句对。实验方案也与(Wu et al., 2016)中的相似：使用newstest2014作为测试集与之前的工作进行比较(Luong et al., 2015a; Zhou et al., 2016; Wu et al., 2016)，而newstest2012和newstest2013的组合被用作开发集。我们还在谷歌的生产英语到法语数据上测试了相同的模型。

表2：WMT'14 En→Fr newstest2014的结果（粗体值代表最好的结果）。

Model	Test Perplexity	Test BLEU	ops/timestep	Total #Parameters	Training Time
MoE with 2048 Experts	2.69	40.35	85M	8.7B	3 days/64 k40s
MoE with 2048 Experts (longer training)	2.63	40.56	85M	8.7B	6 days/64 k40s
GNMT (Wu et al., 2016)	2.79	39.22	214M	278M	6 days/96 k80s
GNMT+RL (Wu et al., 2016)	2.96	39.92	214M	278M	6 days/96 k80s
PBMT (Durrani et al., 2014)		37.0			
LSTM (6-layer) (Luong et al., 2015b)		31.5			
LSTM (6-layer+PosUnk) (Luong et al., 2015b)		33.1			
DeepAtt (Zhou et al., 2016)		37.7			
DeepAtt+PosUnk (Zhou et al., 2016)		39.2			

表3：WMT'14 En → De newstest2014的结果（粗体值代表最佳结果）。

Model	Test Perplexity	Test BLEU	ops/timestep	Total #Parameters	Training Time
MoE with 2048 Experts	4.64	26.03	85M	8.7B	1 day/64 k40s
GNMT (Wu et al., 2016)	5.25	24.91	214M	278M	1 day/96 k80s
GNMT+RL (Wu et al., 2016)	8.08	24.66	214M	278M	1 day/96 k80s
PBMT (Durrani et al., 2014)		20.7			
DeepAtt (Zhou et al., 2016)		20.6			

表4：Google Production En→Fr数据集的结果（粗体值代表最佳结果）。

Model	Eval Perplexity	Eval BLEU	Test Perplexity	Test BLEU	ops/timestep	Total #Parameters	Training Time
MoE with 2048 Experts	2.60	37.27	2.69	36.57	85M	8.7B	1 day/64 k40s
GNMT (Wu et al., 2016)	2.78	35.80	2.87	35.56	214M	278M	6 days/96 k80s

结果：表 2、表 3 和表 4 显示了我们最大模型的结果与已发布结果的比较。我们的方法在 WMT’14 En→Fr 和 En→De 基准测试中获得了 40.56 和 26.03 的 BLEU 分数。由于我们的模型没有使用 RL 细化，这些结果在强基线之上构成了 1.34 和 1.12 BLEU 分数的显着提升（Wu 等人，2016 年）。困惑分数也更好。² 在 Google Production 数据集上，即使只训练了六分之一的时间，我们的模型也获得了 1.01 的高测试 BLEU 分数。

5.4 多语言机器翻译

数据集：(Johnson et al., 2016) 在十二种语言对的非常大的组合数据集上训练单个 GNMT (Wu et al., 2016) 模型。结果比 12 个单独训练的单对 GNMT 模型差一些。这并不奇怪，因为十二个模型的容量是一个模型的 12 倍，总训练量是一个模型的 12 倍。我们用一个模型重复这个实验。有关模型架构的详细信息，请参阅附录 E。我们在与 (Johnson et al., 2016) 相同的数据集上训练我们的模型，并处理相同数量的训练示例（约 30 亿句对）。由于我们模型的计算预算较低，我们的训练时间较短。

结果：表 5 给出了单对 GNMT 模型、多语言 GNMT 模型和多语言 MoE 模型的结果。MoE 模型在开发集上的困惑度比多语言 GNMT 模型低 19%。在 BLEU 分数上，MoE 模型在 12 个语言对中的 11 个上显着击败多语言 GNMT 模型（高达 5.84 分），甚至在 12 个语言对中的 8 个上击败单语言 GNMT 模型。英语→韩语的糟糕表现似乎是严重过度训练的结果，至于罕见的语言对，训练语料库中的少量真实示例被高度过度采样。

Table 5: Multilingual Machine Translation (bold values represent best results).

	GNMT-Mono	GNMT-Multi	MoE-Multi	MoE-Multi vs. GNMT-Multi
Parameters ops/timestep	278M / model 212M various	278M 212M	8.7B 102M	
training time, hardware		21 days, 96 k20s	12 days, 64 k40s	
Perplexity (dev)		4.14	3.35	-19%
French → English Test BLEU	36.47	34.40	37.46	+3.06
German → English Test BLEU	31.77	31.17	34.80	+3.63
Japanese → English Test BLEU	23.41	21.62	25.91	+4.29
Korean → English Test BLEU	25.42	22.87	28.71	+5.84
Portuguese → English Test BLEU	44.40	42.53	46.13	+3.60
Spanish → English Test BLEU	38.00	36.04	39.39	+3.35
English → French Test BLEU	35.37	34.00	36.59	+2.59
English → German Test BLEU	26.43	23.15	24.53	+1.38
English → Japanese Test BLEU	23.66	21.10	22.78	+1.68
English → Korean Test BLEU	19.75	18.41	16.62	-1.79
English → Portuguese Test BLEU	38.40	37.35	37.90	+0.55
English → Spanish Test BLEU	34.50	34.25	36.21	+1.96

六，结论

这项工作首次证明了深度网络中条件计算的重大胜利。我们仔细确定了条件计算的设计注意事项和挑战，并结合算法和工程解决方案来解决这些问题。虽然我们专注于文本，但如果提供足够大的训练集，条件计算也可能对其他领域有所帮助。我们期待在未来几年看到条件计算的许多新颖实现和应用。

ACKNOWLEDGMENTS

我们要感谢所有帮助我们完成这个项目的 Google Brain 和 Google Translate 团队成员，特别是 Zhifeng Chen、Yonghui Wu 和 Melvin Johnson。还要感谢我们的匿名 ICLR 审稿人，他们为改进本文提出了有益的建议。

² 报告了与我们的模型和 GNMT 使用的标记化相关的困惑。

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016. URL <http://arxiv.org/abs/1603.04467>

Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. *CoRR*, abs/1611.06194, 2016. URL <http://arxiv.org/abs/1611.06194>

A. Almahairi, N. Ballas, T. Cooijmans, Y. Zheng, H. Larochelle, and A. Courville. Dynamic Capacity Networks. *ArXiv e-prints*, November 2015.

Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*, 2015.

Dzmitry Bahdanau、Kyunghyun Cho 和 Yoshua Bengio。通过联合学习对齐和翻译进行神经机器翻译。arXiv 预印本 arXiv:1409.0473, 2014。

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

Yoshua Bengio、Nicholas Léonard 和 Aaron Courville。通过随机神经元估计或传播梯度以进行条件计算。arXiv 预印本 arXiv:1308.3432, 2013。

Ciprian Chelba、Tomas Mikolov、Mike Schuster、Qi Ge、Thorsten Brants、Phillipp Koehn 和 Tony Robinson。用于衡量统计语言建模进展的十亿字基准。arXiv 预印本 arXiv:1312.3005, 2013。

K. Cho 和 Y. Bengio。在深度学习中以指数方式增加条件计算的容量与计算比。ArXiv 电子版, 2014 年 6 月。

Ronan Collobert、Samy Bengio 和 Yoshua Bengio。用于超大规模问题的 SVM 的并行混合。神经计算, 2002。

安德鲁·戴维斯和伊塔马尔·阿雷尔。深度神经网络中条件前馈计算的低秩近似。arXiv 预印本 arXiv:1312.4461, 2013。

Marc Peter Deisenroth 和 Jun Wei Ng。分布式高斯过程。在 ICML, 2015 年。

John Duchi、Elad Hazan 和 Yoram Singer。用于在线学习和随机优化的自适应子梯度方法, 2010 年。

Nadir Durrani、Barry Haddow、Philipp Koehn 和 Kenneth Heafield。爱丁堡基于短语的 wmt-14 机器翻译系统。在第九届统计机器翻译研讨会论文集中, 2014 年。

David Eigen、Marc' Aurelio Ranzato 和 Ilya Sutskever。在专家的深度混合中学习因式表示。arXiv 预印本 arXiv:1312.4314, 2013。

Ekaterina Garmash and Christof Monz. Ensemble learning for multi-source neural machine translation. In *staff.science.uva.nl/c.monz*, 2016.

Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 2000.

Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. Memory-efficient backpropagation through time. *CoRR*, abs/1606.03401, 2016. URL <http://arxiv.org/abs/1606.03401>

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath 等。用于语音识别声学建模的深度神经网络：四个研究小组的共同观点。 IEEE 信号处理杂志, 2012 年。

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computing*, 1991.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda B. Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google’s multilingual neural machine translation system: Enabling zero-shot translation. *CoRR*, abs/1611.04558, 2016. URL <http://arxiv.org/abs/1611.04558>

Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computing*, 1994.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Reinhard Kneser and Hermann Ney. Improved backoff for m-gram language modeling., 1995.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Quoc V. Le, Marc’Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeffrey Dean, and Andrew Y. Ng. Building high-level features using large scale unsupervised learning. In *ICML*, 2012.

Patrick Gallinari Ludovic Denoyer. Deep sequential neural network. *arXiv preprint arXiv:1410.0510*, 2014.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *EMNLP*, 2015a.

Minh-Thang Luong, Ilya Sutskever, Quoc V. Le, Oriol Vinyals, and Wojciech Zaremba. Addressing the rare word problem in neural machine translation. *ACL*, 2015b.

Carl Edward Rasmussen and Zoubin Ghahramani. Infinite mixtures of Gaussian process experts. *NIPS*, 2002.

Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH*, pp. 338–342, 2014.

Mike Schuster and Kaisuke Nakajima. Japanese and Korean voice search. *ICASSP*, 2012.

Babak Shahbaba and Radford Neal. Nonlinear models using dirichlet process mixtures. *JMLR*, 2009.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Lucas Theis and Matthias Bethge. Generative image modeling using spatial LSTMs. In *NIPS*, 2015.

Volker Tresp. Mixtures of Gaussian Processes. In *NIPS*, 2001.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes 和 Jeffrey Dean. 谷歌的神经机器翻译系统：弥合人类和机器翻译之间的差距。arXiv 预印本 arXiv:1609.08144, 2016.

Bangpeng Yao, Dirk Walther, Diane Beck, and Li Fei-fei. Hierarchical mixture of classification experts uncovers interactions between brain regions. In *NIPS*. 2009.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, and Wei Xu. Deep recurrent models with fast-forward connections for neural machine translation. *arXiv preprint arXiv:1606.04199*, 2016.

APPENDICES

负载平衡损失

正如第 4 节中所讨论的，出于负载平衡的目的，我们想要定义一个额外的损失函数来鼓励专家接收大致相等数量的训练示例。不幸的是，专家收到的样本数量是离散量，因此不能用于反向传播。相反，我们定义了一个平滑的估计器 $\text{Load}(X)$ ，它是为输入的批次 X 分配给每个专家的示例数。平滑度允许我们通过估计器反向传播梯度。这就是门函数中噪声项的用途。我们将 $P(x, i)$ 定义为 $G(x)$ 中不为零的概率，给定元素 i 上的新随机噪声选择，但保留其他元素上已采样的噪声选择。为了计算 $P(x, i)$ ，我们注意到当且仅当 $H(x)_i$ 大于 $H(x)$ 的第 k 个最大元素（不包括它自身）时， $G(x)_i$ 是非零的。概率计算为：

$$\begin{aligned} P(x, i) = \Pr & \left((x \cdot W_g)_i + \text{StandardNormal}() \cdot \text{Softplus}((x \cdot W_{noise})_i) \right. \\ & \left. > \text{kth_excluding}(H(x), k, i) \right) \end{aligned} \quad (8)$$

其中 $\text{kth_excluding}(v, k, i)$ 表示 v 的第 k 个最高分量，不包括分量 i 。简化，我们得到：

$$P(x, i) = \Phi \left(\frac{(x \cdot W_g)_i - \text{kth_excluding}(H(x), k, i)}{\text{Softplus}((x \cdot W_{noise})_i)} \right) \quad (9)$$

其中 Φ 是标准正态分布的 CDF。

$$\text{Load}(X)_i = \sum_{x \in X} P(x, i) \quad (10)$$

我们现在可以将负载损耗定义为负载向量变异系数的平方乘以手动调整的比例因子 w 负载。

$$L_{load}(X) = w_{load} \cdot CV(\text{Load}(X))^2 \quad (11)$$

Initial Load Imbalance: 为了避免内存不足错误，我们需要在专家负载大致相等的状态下初始化网络（因为软约束需要一些时间才能起作用）。为此，我们将矩阵 W_g 和 W_{noise} 初始化为全零，这不会产生信号和一些噪声。

实验：我们使用 $w_{importance}$ 和 w_{load} 的不同值训练了一组具有相同架构的模型（附录 C 中描述的 MoE-256 模型）。我们对每个模型进行了 10 个时期的训练，然后测量了测试集上的困惑度。我们还测量了重要性和负载的变异系数，以及负载最重的专家与平均负载的比率。最后一个值对于分布式硬件上的负载平衡目的很重要。所有这些指标都在几个训练批次中取平均值。

表 6：不同损失组合的实验。

$w_{importance}$	w_{load}	Test Perplexity	$CV(\text{Importance}(X))$	$CV(\text{Load}(X))$	$\frac{\max(\text{Load}(X))}{\text{mean}(\text{Load}(X))}$
0.0	0.0	39.8	3.04	3.01	17.80
0.2	0.0	35.6	0.06	0.17	1.47
0.0	0.2	35.7	0.22	0.04	1.15
0.1	0.1	35.6	0.06	0.05	1.14
0.01	0.01	35.7	0.48	0.11	1.37
1.0	1.0	35.7	0.03	0.02	1.07

结果：结果报告在 nTableuality, e6.wh 中，所有至少包含两种损失之一的组合导致模型质量非常相似，而没有损失的模型质量要差得多。wload 值较高的模型对超载最重的 expert 的负载较低。

B 专家的等级组合

如果专家数量非常多，我们可以通过使用两级分层 MoE 来减少分支因子。在分层 MoE 中，初级门控网络选择“专家”的稀疏加权组合，每个“专家”本身都是具有自己的门控 bnientwatoiorkn. 3ofIf “ingnetwork”的专家的次级混合，分层 MoE 由一组 b 每个专家，我们用 G 表示初级门控网络，用 $(G, 1 \dots G), 2a$ 表示初级门控网络，用 $(E, 0, 0 \dots E), 0$ 表示专家网络。 $, la, b$ MoE 的输出由下式给出：

$$y_H = \sum_{i=1}^a \sum_{j=1}^b G_{primary}(x)_i \cdot G_i(x)_j \cdot E_{i,j}(x) \quad (12)$$

我们的专家利用率指标更改为以下内容：

$$Importance_H(X)_{i,j} = \sum_{x \in X} G_{primary}(x)_i \cdot G_i(x)_j \quad (13)$$

负载 $(X)_{Hi, j} = (X)_i \cdot L|X(i)|^{Load_{primary}} \cdot L|X(i)|^{Load_i}$ ¹⁴⁾ 和 Load_i 表示主门控网络和第 i 个门控网络的负载函数

次级门控网络分别。 $X(i)$ 表示 $G(x)_{primaryi > 0}$ 的 X 的子集。

It would seem simpler to let $Load_H(X)_{i,j} = Load_i(X_i)_j$, but this would not have a gradient with respect to the primary gating network, so we use the formulation above.

C 10 亿字语言建模基准 - 实验细节

C.1 8-MILLION-OPERATIONS-PER-TIMESTEP MODELS

模型架构：我们的模型由五层组成：词嵌入层、循环长短期记忆（LSTM）层（Hochreiter & Schmidhuber, 1997）、MoE 层、第二个 LSTM 层和一个 softmax 层。嵌入层的维数、每个 LSTM 层的单元数以及 MoE 层的输入和输出维数都等于 512。对于除 softmax 之外的每一层，我们应用 dropout¹⁴⁾ 到层输出，以概率 DropProb 丢弃每个激活，否则除以 $1 - DropProb$ 。dropout 之后，将上一层的输出添加到该层的输出中。

This residual connection encourages gradient flow (He et al., 2015).

MoE 层架构：MoE 层中的每个专家都是一个前馈网络，具有一个大小为 1024 的 ReLU 激活隐藏层和一个大小为 512 的输出层。因此，每个专家包含 $[512 * 1024] + [1024 * 512] = 1M$ 参数。MoE 层的输出在 dropout 之前通过一个 sigmoid 函数传递。我们改变模型之间的专家数量，使用具有 4、32 和 256 位专家的普通 MoE 层和具有 256、1024 和 4096 位专家的分层 MoE 层。我们将生成的模型称为 MoE-4、MoE-32、MoE-256、MoE-256-h、MoE-1024-h 和 MoE-4096-h。对于分层的 MoE 层，第一级分支因子为 16，对应于我们集群中的 GPU 数量。我们使用 Noisy-Top-K Gating（参见 oSrewctaiosn162,.1c）oElayers.Th，其中 k = 4 用于普通 MoE 层，k = 2 用于分层 MoE 层的每一层。因此，每个示例恰好由 4 位专家处理，总共 4M 操作/时间步。两个 LSTM 层各贡献 2M ops/timestep，总计 8M。

¹⁴⁾ We have not found the need for deeper hierarchies.

Computationally-Matched Baselines: MoE-4 模型不使用稀疏性，因为总是使用所有 4 个专家。此外，我们还训练了四个计算匹配的无稀疏性基线模型：

- MoE-1-Wide: MoE 层由单个“专家”组成，其中包含一个大小为 4096 的 ReLU 激活隐藏层。
- MoE-1-Deep: The MoE layer consists of a single "expert" containing four ReLU-activated hidden layers, each with size 1024.
- 4xLSTM-512: 我们用两个额外的 512 单元 LSTM 层替换了 MoE 层。
- LSTM-2048-512: 该模型包含一个 2048 单元的 LSTM 层（没有 MoE）。LSTM 的输出被预测到 512 维 (Sak et al., 2014)。LSTM 的下一个时间步接收投影输出。这与 (Jozefowicz et al., 2016) 中发布的模型之一相同。我们重新运行它以解决训练方案的差异，并获得与已发布的结果非常相似的结果。

训练：使用第 3 节中描述的同步方法在 16 个 K40 GPU 集群上训练模型。每批包含一组句子，总计大约 300,000 个单词。为了节省时间，我们将训练限制在 10 个时期 (27,000 步)。所有模型的训练都花费了 12–16 小时，MoE-4 除外，它花费了 18 小时（因为所有专家计算仅在 16 个 GPU 中的 4 个上执行）。我们使用了 Adam 优化器 (Kingma & Ba, 2015)。基础学习率在前 1000 步训练时线性增加，之后下降，与步数的平方根反比成正比。与 (Jozefowicz et al., 2016) 中的模型类似，使用重要性采样对 Softmax 输出层进行了有效训练。对于每个模型，我们执行了超参数搜索以找到最佳丢失概率，增量为 0.1。

To ensure balanced expert utilization we set $w_{importance} = 0.1$ and $w_{load} = 0.1$, as described in Section 4 and Appendix A

结果：我们在 holdout 数据集上使用 perplexity 来评估我们的模型，该数据集被 (Chelba et al., 2013; Jozefowicz et al., 2016) 使用。我们遵循标准程序并对所有单词求和，包括句末符号。结果报告在表 7 中。对于每个模型，我们报告了测试困惑度、计算预算、参数计数、DropP rob 的值和计算效率。

Table 7: Model comparison on 1 Billion Word Language Modeling Benchmark. Models marked with * are from (Jozefowicz et al., 2016).

Model	Test Perplexity 10 epochs	Test Perplexity (final)	ops/timestep (millions)	#Params excluding embed. & softmax (millions)	Total #Params (billions)	Drop- Prob	TFLOPS per GPU (observed)
Kneser-Ney 5-gram*		67.6	0.00001		1.8		
LSTM-512-512*		54.1	2.4	2.4	0.8	0.1	
LSTM-1024-512*		48.2	4.7	4.7	0.8	0.1	
LSTM-2048-512*	45.0	43.7	9.4	9.4	0.8	0.1	0.61
LSTM-2048-512	44.7		9.4	9.4	0.8	0.1	1.21
4xLSTM-512	46.0		8.4	8.4	0.8	0.1	1.07
MoE-1-Wide	46.1		8.4	8.4	0.8	0.1	1.29
MoE-1-Deep	45.7		8.4	8.4	0.8	0.1	1.29
MoE-4	45.0		8.4	8.4	0.8	0.1	0.52
MoE-32	39.7		8.4	37.8	0.9	0.1	0.87
MoE-256	35.7		8.6	272.9	1.1	0.1	0.81
MoE-256-h	36.0		8.4	272.9	1.1	0.1	0.89
MoE-1024-h	34.6		8.5	1079.0	1.9	0.2	0.90
MoE-4096-h	34.1		8.9	4303.4	5.1	0.2	0.74
2xLSTM-8192-1024*	34.7	30.6	151.0	151.0	1.8	0.25	1.09
MoE-34M	31.3		33.8	4313.9	6.0	0.3	1.22
MoE-143M	28.0		142.7	4371.1	6.0	0.4	1.56

C.2 更昂贵的模型

我们运行了两个额外的模型 (MoE-34M 和 MoE-143M) 来研究在存在大型 MoE 层的情况下增加更多计算的效果。这些模型的计算预算为 34M 和 143M ops/timestep。与上述模型类似，这些模型在两个 LSTM 层之间使用了一个 MoE 层。嵌入层的维数以及 MoE 层的输入和输出维数设置为 1024 而不是 512。对于 MoE-34M, LSTM 层有 1024 个单元。对于 MoE-143M, LSTM 层有 4096 个单元和大小为 1024 的输出投影 (Sak 等人, 2014)。MoE-34M 使用具有 1024 个专家的分层 MoE 层，每个专家都有一个大小为 2048 的隐藏层。MoE-143M 使用具有 256 个专家的分层 MoE 层，每个专家都有一个大小为 8192 的隐藏层。两个模型在 MoE 中都有 4B 个参数层。我们为每个模型搜索了最好的 DropP rob，并对每个模型进行了 10 个 epoch 的训练。

这两个模型分别实现了 31.3 和 28.0 的测试困惑度，表明即使存在较大的 MoE，更多的计算仍然有用。结果报告在表 7 的底部。两个模型中较大的一个与文献中发布的最佳模型具有相似的计算预算，并且训练时间相似。在 10 个 epoch 之后进行比较，我们的模型的测试困惑度降低了 18%。

D 1000 亿字的谷歌新闻语料库 - 实验细节

模型架构：这些模型在结构上类似于上一节中描述的每时间步长 800 万次操作的模型。我们改变模型之间的专家数量，使用具有 32 名专家的普通 MoE 层和具有 256、1024、4096、16384、65536 和 131072 名专家的分层 MoE 层。对于分层的 MoE 层，第一级分支因子分别为 32、32、64、128、256 和 256。

训练：模型在 32 个 Tesla K40 GPU 集群上训练，除了最后两个模型，它们在 64 和 128 个 GPU 集群上训练，以便为所有参数提供足够的内存。对于所有模型，训练批量大小约为 250 万个单词。模型经过大约 1000 亿个单词的一次性训练。

我们实施了多项内存优化，以适应每个 GPU 多达 10 亿个参数。首先，我们不存储专家隐藏层的激活，而是在向后传递时重新计算它们。其次，我们修改专家参数的优化器以减少辅助存储：

Adam 优化器 (Kingma & Ba, 2015) 保留每个参数梯度的一阶和二阶矩估计。这使所需内存增加了三倍。为避免保留一阶矩估计量，我们设置 $\beta_1 = 0$ 。为了减小二阶矩估计量的大小，我们将其替换为因式近似。对于参数矩阵，我们不是维护一个完整的二次矩估计矩阵，而是维护该矩阵的行方向和列方向平均值的向量。在每一步，估计矩阵被认为是这两个向量除以任一向量的平均值的外积。这种技术可以类似地应用于 Adagrad (Duchi et al., 2010)。

表 8: 1000 亿字 Google 新闻数据集的模型比较

Model	Test Perplexity .1 epochs	Test Perplexity 1 epoch	ops/timestep (millions)	#Params excluding embed. & softmax (millions)	Total #Params (billions)	TFLOPS per GPU (observed)
Kneser-Ney 5-gram	67.1	45.3	0.00001		76.0	
4xLSTM-512	54.5	47.0	8.4	8.4	0.1	1.23
MoE-32	48.5	40.4	8.4	37.8	0.1	0.83
MoE-256-h	42.8	35.3	8.4	272.9	0.4	1.11
MoE-1024-h	40.3	32.7	8.5	1079.0	1.2	1.14
MoE-4096-h	38.9	30.9	8.6	4303.4	4.4	1.07
MoE-16384-h	38.2	29.7	8.8	17201.0	17.3	0.96
MoE-65536-h	38.2	28.9	9.2	68791.0	68.9	0.72
MoE-131072-h	39.8	29.2	9.7	137577.6	137.7	0.30

结果：我们在保留数据集上使用困惑度来评估我们的模型。结果如表 8 所示。对于 680 亿参数的 MoE，经过 1000 亿训练词后的困惑度降低了 39%

模型比基线模型。值得注意的是，与其他模型相比，最大模型（0.30 TFLOPS/GPU）的计算效率非常低。这可能是因为为了与其他模型进行比较，我们没有根据 GPU 的数量按比例增加训练批次大小。为了进行比较，我们包括了由 4 个 LSTM 组成的计算匹配基线模型的结果，以及具有 Kneser–Ney 平滑的未修剪 5-gram 模型的结果 (Kneser & Ney, 1995)。⁴

4个

电子机器翻译 - 实验细节

单语言对 MoE 模型的模型架构：我们的模型是 (Wu et al., 2016) 中描述的 GNMT 模型的修改版本。为了减少计算量，我们将编码器和解码器中的 LSTM 层数分别从 9 层和 8 层减少到 3 层和 2 层。我们在编码器（第 2 层和第 3 层之间）和解码器（第 1 层和第 2 层之间）中插入 MoE 层。我们在编码器和解码器之间使用注意力机制，第一个解码器 LSTM 从注意力 5 接收输出并提供输入。我们模型中的所有层的输入和输出维度均为 512。我们的 LSTM 层有 2048 个隐藏单元，输出投影为 512 维。我们在所有 LSTM 和 MoE 层周围添加残差连接以鼓励梯度流 (He et al., 2015)。与 GNMT 类似，为了有效处理稀有词，我们在系统中使用子词单元（也称为“wordpieces”）(Schuster & Nakajima, 2012) 作为输入和输出。

We use a shared source and target vocabulary of 32K wordpieces. We also used the same beam search technique as proposed in (Wu et al. 2016).

我们在 MoE 层中使用不同数量的专家训练模型。除了没有 MoE 层的基线模型外，我们还训练了包含 32 名专家的平面 MoE 层模型，以及包含 512 名和 2048 名专家的分层 MoE 层模型。平面 MoE 层使用 $k = 4$ ，分层 MoE 模型在门控网络的每一层使用 $k = 2$ 。因此，每个输入都由每个 MoE 层中的 4 个专家处理。MoE 层中的每个专家都是一个前馈网络，具有一个大小为 2048 的隐藏层和 ReLU 激活。因此，每个专家包含 $[512 * 2048] + [2048 * 512] = 2M$ 个参数。MoE 层的输出通过 sigmoid 函数传递。我们使用附录 F 中描述的严格平衡门函数。

多语言 MoE 模型的模型架构：我们使用与单语言对模型相同的模型架构，但有以下例外：我们使用第 2.1 节中描述的 noisy-top-k 门控，而不是附录 F 中的方案。编码器和解码器中的 MoE 层是非分层的 MoE， $n = 512$ 个专家， $k = 2$ 。每个专家都有一个更大的隐藏层，大小为 8192。这使 MoE 层的计算量加倍，提高了计算预算整个模型从 85M 到 102M ops/timestep。

训练：我们使用 Adam 优化器训练我们的网络 (Kingma & Ba, 2015)。基础学习率在前 2000 个训练步骤中线性增加，在另外 8000 个步骤中保持不变，然后降低，以便与步骤数的平方根反比成正比。对于单语言对模型，与 (Wu et al., 2016) 类似，我们将 dropout (Zaremba et al., 2014) 应用于所有嵌入、LSTM 和 MoE 层的输出，使用 DropP rob = 0.4。如第 3 节所述，训练在多达 64 个 GPU 的集群上同步完成。每个训练批次由一组句子对组成，每个 GPU 包含大约 16000 个单词。

To ensure balanced expert utilization we set $w_{importance} = 0.01$ and $w_{load} = 0.01$, as described in Section 4 and Appendix A

Metrics: We evaluated our models using the perplexity and the standard BLEU score metric. We reported tokenized BLEU score as computed by the multi-bleu.pl script, downloaded from the public implementation of Moses (on Github), which was also used in (Luong et al. 2015a).

⁴ 虽然语料库的原始大小为 1300 亿个单词，但神经模型的训练最多可容纳 1000 亿个单词。报告的 Kneser–Ney 5-gram 模型分别接受了超过 130 亿和 1300 亿个单词的训练，使它们比其他报告的结果略有优势。

⁵ 出于性能原因，我们使用了与 (Wu 等人, 2016 年) 中描述的注意力函数略有不同的函数 - 请参阅附录 G

结果：第 5.3 节中的表 2、表 3 和表 4 显示了我们的结果与其他已发表方法的比较。图 4 显示了测试困惑度作为（训练数据的）源句子中单词数量的函数，这些句子是为具有不同专家数量的模型处理的。从图中可以看出，随着我们将专家数量增加到接近 2048，我们模型的测试困惑度不断提高。

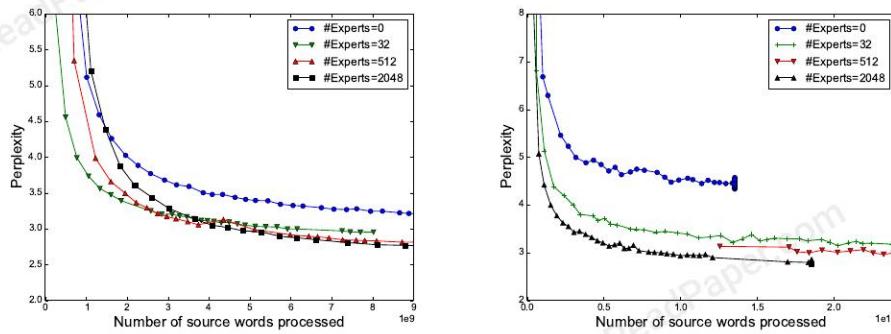


图 4：WMT’14 En→Fr（左）和 Google Production En→Fr（右）数据集的困惑度作为处理的单词数量的函数。训练开始时模型之间的巨大差异是由于批大小不同造成的。除没有专家的模型外，所有模型都会产生相同的计算预算（8500 万次操作/时间步长）。

我们发现专家确实通过句法和/或语义变得高度专业化，如表 9 所示。例如，当不定冠词“a”在表示重要性或领导力的动词短语中引入直接宾语时，使用一位专家。

表 9：与 WMT’14 En→Fr 翻译模型的编码器部分的 MoE 层中的一些 2048 位专家对应的上下文。对于每个专家 i ，我们按 $G(x)_i$ 的降序对训练批次中的输入进行排序，并显示输入句子中相应位置周围的单词。

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 swiftly ...
... the generation of play a leading to drastically ...
... 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 ...
...

F 严格平衡的门控

由于我们的基础设施中的一些特性已经得到修复，在我们运行一些机器翻译实验时，如果每个专家都接收到完全相同的批量大小，我们的模型运行得更快。为了适应这一点，我们使用了下面描述的不同门控功能。

回想一下，我们将 softmax 门函数定义为：

$$G_\sigma(x) = \text{Softmax}(x \cdot W_g) \quad (15)$$

Sparse Gating (alternate formulation): To obtain a sparse gating vector, we multiply $G_\sigma(x)$ component-wise with a sparse mask $M(G_\sigma(x))$ and normalize the output. The mask itself is a function of $G_\sigma(x)$ and specifies which experts are assigned to each input example:

$$G(x)_i = \frac{G_\sigma(x)_i M(G_\sigma(x))_i}{\sum_{j=1}^n G_\sigma(x)_j M(G_\sigma(x))_j} \quad (16)$$

Top-K Mask: 为了在此公式中实施 top-k 门控，我们将令 $M(v) = \text{TopK}(v, k)$ ，其中：

$$\text{TopK}(v, k)_i = \begin{cases} 1 & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ 0 & \text{otherwise.} \end{cases} \quad (17)$$

Batchwise Mask: 为了强制每个专家接收完全相同数量的示例，我们引入了另一种掩码函数 $M(X, m)$ ，它对输入向量的批次进行分批操作。我们不是保留每个示例的前 k 个值，而是保留每个专家的前 m 个值 training batch, where $m = \frac{k|X|}{n}$, so that each example is sent to an average of k experts.

$$M_{batchwise}(X, m)_{j,i} = \begin{cases} 1 & \text{if } X_{j,i} \text{ is in the top } m \text{ values for to expert } i \\ 0 & \text{otherwise} \end{cases} \quad (18)$$

As our experiments suggest and also observed in [Ioffe & Szegedy, 2015], using a batchwise function when we may have few examples, the batch processing (e.g. M) needs to be modified. Our solution is to train each expert threshold vector T to近似分批掩码的效果。我们在推理时使用以下掩码：

$$M_{threshold}(x, T)_i = \begin{cases} 1 & \text{if } x_i > T_i \\ 0 & \text{otherwise} \end{cases} \quad (19)$$

为了学习阈值，我们在训练时应用了一个额外的损失，当批量掩码和阈值掩码相同时，该损失最小化。

$$L_{batchwise}(X, T, m) = \sum_{i=1}^{|X|} \sum_{j=1}^n (M_{threshold}(x, T)_i - M_{batchwise}(X, m)_{j,i})(X_{j,i} - T_i) \quad (20)$$

$i=1$ G 注意函数

GNMT T(Wuxan uetnda tal., 2016) 中描述的注意力机制涉及一个学习的“注意力函数” $A(x, y)$ i, j ，它采用“源向量” x_i 和“目标向量” y_j ，并且必须是为每个源时间步长 i 和目标时间步长 j 计算。在 GNMT 中，注意力函数被实现为具有大小为 n 的隐藏层的前馈神经网络。它可以表示为：

$$A_{GNMT}(x_i, y_j) = \sum_{d=1}^n V_d \tanh((x_i U)_d + (y_j W)_d) \quad (21)$$

其中 U 和 W 是可训练的权重矩阵， V 是可训练的权重向量。

出于性能原因，在我们的模型中，我们使用了稍微不同的注意力函数：

$$A(x_i, y_j) = \sum_{d=1}^n V_d \tanh((x_i U)_d) \tanh((y_j W)_d) \quad (22)$$

使用我们的注意力函数，我们可以使用优化的矩阵乘法同时计算多个源时间步长和多个目标时间步长的注意力函数。我们发现这两个功能之间的质量差异不大。