OUTRAGEOUSLY LARGE NEURAL NETWORKS: THE SPARSELY-GATED MIXTURE-OF-EXPERTS LAYER

Noam Shazeer[1], Azalia Mirhoseini*†[1], Krzysztof Maziarz*[2], Andy Davis[1], Quoc Le[1], Geoffrey Hinton[1] and Jeff Dean[1]

[1]Google Brain, {noam,azalia,andydavis,qvl,geoffhinton,jeff}@google.com
[2]Jagiellonian University, Cracow, krzysztof.maziarz@student.uj.edu.pl

ABSTRACT

The capacity of a neural network to absorb information is limited by its number of parameters. Conditional computation, where parts of the network are active on a per-example basis, has been proposed in theory as a way of dramatically increasing model capacity without a proportional increase in computation. In practice, however, there are significant algorithmic and performance challenges. In this work, we address these challenges and finally realize the promise of conditional computation, achieving greater than 1000x improvements in model capacity with only minor losses in computational efficiency on modern GPU clusters. We introduce a Sparsely-Gated Mixture-of-Experts layer (MoE), consisting of up to thousands of feed-forward sub-networks. A trainable gating network determines a sparse combination of these experts to use for each example. We apply the MoE to the tasks of language modeling and machine translation, where model capacity is critical for absorbing the vast quantities of knowledge available in the training corpora. We present model architectures in which a MoE with up to 137 billion parameters is applied convolutionally between stacked LSTM layers. On large language modeling and machine translation benchmarks, these models achieve significantly better results than state-of-the-art at lower computational cost.

1 INTRODUCTION AND RELATED WORK

1.1 CONDITIONAL COMPUTATION

Exploiting scale in both training data and model size has been central to the success of deep learning. When datasets are sufficiently large, increasing the capacity (number of parameters) of neural networks can give much better prediction accuracy. This has been shown in domains such as text (Sutskever et al., 2014; Bahdanau et al., 2014; Jozefowicz et al., 2016; Wu et al., 2016), images (Krizhevsky et al., 2012; Le et al., 2012), and audio (Hinton et al., 2012; Amodei et al., 2015). For typical deep learning models, where the entire model is activated for every example, this leads to a roughly quadratic blow-up in training costs, as both the model size and the number of training examples increase. Unfortunately, the advances in computing power and distributed computation fall short of meeting such demand.

Various forms of conditional computation have been proposed as a way to increase model capacity without a proportional increase in computational costs (Davis & Arel, 2013; Bengio et al., 2013; Eigen et al., 2013; Ludovic Denoyer, 2014; Cho & Bengio, 2014; Bengio et al., 2015; Almahairi et al., 2015). In these schemes, large parts of a network are active or inactive on a per-example basis. The gating decisions may be binary or sparse and continuous, stochastic or deterministic. Various forms of reinforcement learning and back-propagation are proposed for trarining the gating decisions.

*Equally major contributors
†Work done as a member of the Google Brain Residency program (g.co/brainresidency)

arXiv:1701.06538v1 [cs.LG] 23 Jan 2017

荒谬地大的神经网络：稀疏门控专家层

Noam Shazeer[1], Azalia Mirhoseini*†[1],Krzysztof Maziarz*[2],Andy Davis[1],QuocLe[1],Geoffrey Hinton[1] and Jeff Dean[1]

[1]Google Brain, {noam,azalia,andydavis,qvl,geoffhinton,jeff}@google.com [2]
雅盖隆大学，克拉科夫，krzysztof.maziarz@student.uj.edu.pl

摘要

神经网络吸收信息的能力受其参数数量的限制。条件计算，即网络的部分按每个示例激活，理论上已被提出作为一种在不成比例增加计算的情况下大幅提高模型容量的方法。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，最终实现了条件计算的承诺，在现代GPU集群上仅略微损失计算效率的情况下，将模型容量提高了1000多倍。我们引入了一个稀疏门控专家层（MoE），由多达数千个前馈子网络组成。一个可训练的门控网络确定对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任务，在这些任务中，模型容量对于吸收训练语料库中可用的海量知识至关重要。我们展示了模型架构，其中MoE被卷积地应用于堆叠的LSTM层之间，最多包含1370亿个参数。在大型语言建模和机器翻译基准测试中，这些模型在更低的计算成本下实现了比当前最佳更好的结果。

1 引言与相关工作

1.1 条件计算

利用训练数据和模型规模的优势一直是深度学习成功的关键。当数据集足够大时，增加神经网络的容量（参数数量）可以显著提高预测精度。这在文本（Sutskever等人，2014年；Bahdanau等人，2014年；Jozefowicz等人，2016年；吴等人，2016年）、图像（Krizhevsky等人，2012年；Le等人，2012年）和音频（Hinton等人，2012年；Amodei等人，2015年）等领域已被证明有效。对于典型的深度学习模型，其中整个模型对每个示例都会被激活，这会导致训练成本大致呈二次方增长，因为模型规模和训练样本数量都在增加。不幸的是，计算能力和分布式计算的进步未能满足这种需求。
各种条件计算方法已被提出，作为一种在不成比例增加计算成本的情况下提高模型容量的方式（Davis & Arel, 2013；Bengio 等人，2013；Eigen 等人，2013；Ludovic Denoyer，2014；Cho & Bengio，2014；Bengio 等人，2015；Almahairi 等人，2015）。在这些方案中，网络的大部分部分基于每个示例是激活的或非激活的。门控决策可以是二元的或稀疏和连续的、随机的或确定的。提出了各种强化学习和反向传播方法来训练门控决策。

*同样重要的贡献者
†作为 Google Brain 住院医师项目成员完成的工作（g.co/brainresidency）

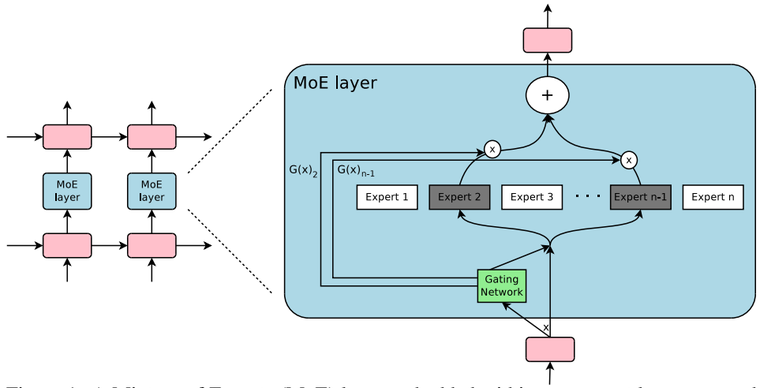

Figure 1: A Mixture of Experts (MoE) layer embedded within a recurrent language model. In this case, the sparse gating function selects two experts to perform computations. Their outputs are modulated by the outputs of the gating network.

While these ideas are promising in theory, no work to date has yet demonstrated massive improvements in model capacity, training time, or model quality. We blame this on a combination of the following challenges:

- Modern computing devices, especially GPUs, are much faster at arithmetic than at branching. Most of the works above recognize this and propose turning on/off large chunks of the network with each gating decision.
- Large batch sizes are critical for performance, as they amortize the costs of parameter transfers and updates. Conditional computation reduces the batch sizes for the conditionally active chunks of the network.
- Network bandwidth can be a bottleneck. A cluster of GPUs may have computational power thousands of times greater than the aggregate inter-device network bandwidth. To be computationally efficient, the relative computational versus network demands of an algorithm must exceed this ratio. Embedding layers, which can be seen as a form of conditional computation, are handicapped by this very problem. Since the embeddings generally need to be sent across the network, the number of (example, parameter) interactions is limited by network bandwidth instead of computational capacity.
- Depending on the scheme, loss terms may be necessary to achieve the desired level of sparsity per-chunk and/or per example. Bengio et al. (2015) use three such terms. These issues can affect both model quality and load-balancing.
- Model capacity is most critical for very large data sets. The existing literature on conditional computation deals with relatively small image recognition data sets consisting of up to 600,000 images. It is hard to imagine that the labels of these images provide a sufficient signal to adequately train a model with millions, let alone billions of parameters.

In this work, we for the first time address all of the above challenges and finally realize the promise of conditional computation. We obtain greater than 1000x improvements in model capacity with only minor losses in computational efficiency and significantly advance the state-of-the-art results on public language modeling and translation data sets.

1.2 Our Approach: The Sparsely-Gated Mixture-of-Experts Layer

Our approach to conditional computation is to introduce a new type of general purpose neural network component: a Sparsely-Gated Mixture-of-Experts Layer (MoE). The MoE consists of a number of experts, each a simple feed-forward neural network, and a trainable gating network which selects a sparse combination of the experts to process each input (see Figure 1). All parts of the network are trained jointly by back-propagation.

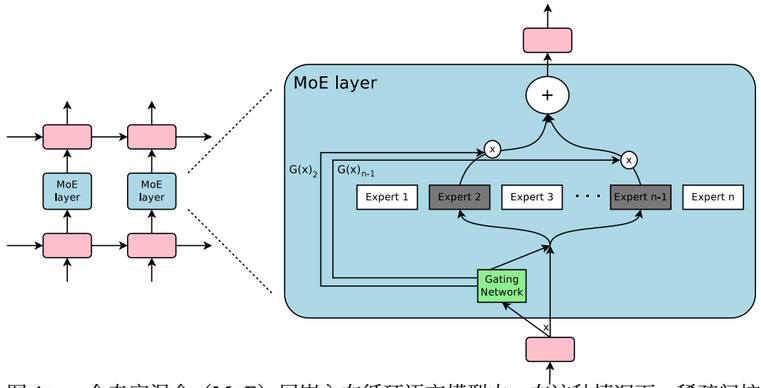

图 1：一个专家混合（MoE）层嵌入在循环语言模型中。在这种情况下，稀疏门控函数选择两个专家执行计算。它们的输出由门控网络的输出调制。

虽然这些想法在理论上很有前景，但迄今为止还没有工作证明在模型容量、训练时间或模型质量方面取得了巨大的改进。我们将此归因于以下挑战的组合：

- 现代计算设备，尤其是 GPU，在算术运算上比分支操作快得多。大多数相关工作都认识到这一点，并提出通过每个门控决策打开或关闭网络的大块。
- 大批量对于性能至关重要，因为它们摊销了参数传输和更新的成本。条件计算减少了网络条件激活块的大批量。
- 网络带宽可能是一个瓶颈。一个 GPU 集群的计算能力可能比设备间网络带宽的总和高出数千倍。为了计算高效，算法的计算需求与网络需求的相对值必须超过这个比率。嵌入层，可以看作是一种条件计算，受此问题的严重限制。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。
- 根据方案不同，可能需要损失项来实现每块和/或每个示例的所需稀疏性。Bengio 等人（2015）使用了三项此类损失。这些问题会影响模型质量和负载均衡。
- 模型容量对于非常大的数据集最为关键。现有关于条件计算的研究文献处理的是相对较小的图像识别数据集，包含最多 60 万张图像。很难想象这些图像的标签能提供足够的信号来充分训练一个具有数百万甚至数十亿参数的模型。

在这项工作中，我们首次解决了所有上述挑战，并最终实现了条件计算的前景。我们获得了超过 1000 倍的模型容量提升，计算效率仅略有下降，并在公共语言建模和翻译数据集上显著推进了当前最佳结果。

1.2 我们的方案：稀疏门控专家层

我们关于条件计算的方法是引入一种新的通用神经网络组件：稀疏门控专家层（MoE）。MoE由多个专家组成，每个专家是一个简单的前馈神经网络，以及一个可训练的门控网络，该门控网络选择稀疏的专家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。

While the introduced technique is generic, in this paper we focus on language modeling and machine translation tasks, which are known to benefit from very large models. In particular, we apply a MoE convolutionally between stacked LSTM layers (Hochreiter & Schmidhuber, 1997), as in Figure 1. The MoE is called once for each position in the text, selecting a potentially different combination of experts at each position. The different experts tend to become highly specialized based on syntax and semantics (see Appendix E Table 9). On both language modeling and machine translation benchmarks, we improve on best published results at a fraction of the computational cost.

1.3 Related work on Mixtures of Experts

Since its introduction more than two decades ago (Jacobs et al., 1991; Jordan & Jacobs, 1994), the mixture-of-experts approach has been the subject of much research. Different types of expert architectures hae been proposed such as SVMs (Collobert et al., 2002), Gaussian Processes (Tresp, 2001; Theis & Bethge, 2015; Deisenroth & Ng, 2015), Dirichlet Processes (Shahbaba & Neal, 2009), and deep networks. Other work has focused on different expert configurations such as a hierarchical structure (Yao et al., 2009), infinite numbers of experts (Rasmussen & Ghahramani, 2002), and adding experts sequentially (Aljundi et al., 2016). Garmash & Monz (2016) suggest an ensemble model in the format of mixture of experts for machine translation. The gating network is trained on a pre-trained ensemble NMT model.

The works above concern top-level mixtures of experts. The mixture of experts is the whole model. Eigen et al. (2013) introduce the idea of using multiple MoEs with their own gating networks as parts of a deep model. It is intuitive that the latter approach is more powerful, since complex problems may contain many sub-problems each requiring different experts. They also allude in their conclusion to the potential to introduce sparsity, turning MoEs into a vehicle for computational computation.

Our work builds on this use of MoEs as a general purpose neural network component. While Eigen et al. (2013) uses two stacked MoEs allowing for two sets of gating decisions, our convolutional application of the MoE allows for different gating decisions at each position in the text. We also realize sparse gating and demonstrate its use as a practical way to massively increase model capacity.

2 The Structure of the Mixture-of-Experts layer

The Mixture-of-Experts (MoE) layer consists of a set of n "expert networks" $E_1, \cdots, E_n$, and a "gating network" G whose output is a sparse n-dimensional vector. Figure 1 shows an overview of the MoE module. The experts are themselves neural networks, each with their own parameters. Although in principle we only require that the experts accept the same sized inputs and produce the same-sized outputs, in our initial investigations in this paper, we restrict ourselves to the case where the models are feed-forward networks with identical architectures, but with separate parameters.

Let us denote by $G(x)$ and $E_i(x)$ the output of the gating network and the output of the i-th expert network for a given input x. The output y of the MoE module can be written as follows:

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

We save computation based on the sparsity of the output of $G(x)$. Wherever $G(x)_i = 0$, we need not compute $E_i(x)$. In our experiments, we have up to thousands of experts, but only need to evaluate a handful of them for every example. If the number of experts is very large, we can reduce the branching factor by using a two-level hierarchical MoE. In a hierarchical MoE, a primary gating network chooses a sparse weighted combination of "experts", each of which is itself a secondary mixture-of-experts with its own gating network. In the following we focus on ordinary MoEs. We provide more details on hierarchical MoEs in Appendix B.

Our implementation is related to other models of conditional computation. A MoE whose experts are simple weight matrices is similar to the parameterized weight matrix proposed in (Cho & Bengio, 2014). A MoE whose experts have one hidden layer is similar to the block-wise dropout described in (Bengio et al., 2015), where the dropped-out layer is sandwiched between fully-activated layers.

虽然所引入的技术是通用的，但在本文中我们专注于语言建模和机器翻译任务，这些任务已知能从非常大的模型中获益。特别是，我们在堆叠的LSTM层之间应用MoE卷积（Hochreiter & Schmidhuber, 1997），如图1所示。MoE为文本中的每个位置调用一次，在每个位置选择潜在的不同专家组合。不同的专家倾向于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，我们在计算成本的一小部分上改进了最佳已发表结果。

1.3 专家混合相关工作

自其二十多年前提出以来（Jacobs 等人，1991 年；Jordan & Jacobs，1994 年），专家混合方法一直是研究的热点。提出了不同类型的专家架构，如 SVM（Collobert 等人，2002 年）、高斯过程（Tresp，2001 年；Theis & Bethge，2015 年；Deisenroth & Ng，2015 年）、狄利克雷过程（Shahbaba & Neal，2009 年）和深度网络。其他工作则关注不同的专家配置，如分层结构（Yao 等人，2009 年）、无限数量的专家（Rasmussen & Ghahramani，2002 年）以及依次添加专家（Aljundi 等人，2016 年）。Garmash & Monz（2016 年）提出了一种用于机器翻译的专家混合格式集成模型。门控网络在预训练的集成 NMT 模型上进行训练。

上述工作涉及顶级专家混合。专家混合是整个模型。Eigen 等人 (2013) 引入了使用多个 MoE 并为其配备各自的门控网络作为深度模型一部分的想法。直观地看，后一种方法更强大，因为复杂问题可能包含许多子问题，每个子问题都需要不同的专家。他们在结论中也暗示了引入稀疏性的潜力，将 MoE 变成计算计算的载体。

我们的工作建立在将 MoE 作为通用神经网络组件的使用之上。虽然 Eigen 等人 (2013) 使用两个堆叠的 MoE 允许进行两组门控决策，但我们的卷积应用 MoE 允许在文本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幅增加模型容量的用途。

2 专家混合层的结构

专家混合 (MoE) 层由一组 n "专家网络" $E_1, \cdots, E_n$ 和一个"门控网络" G 组成，其输出是一个稀疏 n-维向量。图 1 展示了 MoE 模块的整体结构。专家本身是神经网络，每个都有自己的参数。虽然原则上我们只需要专家接受相同大小的输入并产生相同大小的输出，但在本文的初步研究中，我们将自己限制在模型是具有相同架构但具有独立参数的前馈网络的情况。

我们用 $G(x)$ 和 $E_i(x)$ 分别表示门控网络的输出以及给定输入 x 时第 i 个专家网络的输出。MoE 模块的输出可以表示如下：

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

我们基于 $G(x)$ 输出的稀疏性来节省计算量。只要 $G(x)_i = 0$，我们就不需要计算 $E_i(x)$。在我们的实验中，我们最多有数千个专家，但每个示例只需要评估其中的一小部分。如果专家数量非常大，我们可以通过使用双层分层 MoE 来降低分支因子。在分层 MoE 中，一个主门控网络选择"专家"的稀疏加权组合，而每个"专家"本身又是一个具有自己门控网络的二级专家混合。在以下内容中，我们专注于普通的 MoE。我们在附录 B 中提供了更多关于分层 MoE 的细节。

我们的实现与其他条件计算模型相关。一个专家为简单权重矩阵的MoE与(Cho & Bengio, 2014)中提出的参数化权重矩阵相似。一个专家具有一层隐藏层的MoE与(Bengio 等人, 2015)中描述的块状dropout相似，其中被丢弃的层位于全激活层之间。

2.1 GATING NETWORK

Softmax Gating: A simple choice of non-sparse gating function (Jordan & Jacobs, 1994) is to multiply the input by a trainable weight matrix W_g and then apply the $Softmax$ function.

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

Noisy Top-K Gating: We add two components to the Softmax gating network: sparsity and noise. Before taking the softmax function, we add tunable Gaussian noise, then keep only the top k values, setting the rest to $-\infty$ (which causes the corresponding gate values to equal 0). The sparsity serves to save computation, as described above. While this form of sparsity creates some theoretically scary discontinuities in the output of gating function, we have not yet observed this to be a problem in practice. The noise term helps with load balancing, as will be discussed in Appendix A. The amount of noise per component is controlled by a second trainable weight matrix W_{noise}.

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

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

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

Training the Gating Network We train the gating network by simple back-propagation, along with the rest of the model. If we choose $k > 1$, the gate values for the top k experts have nonzero derivatives with respect to the weights of the gating network. This type of occasionally-sensitive behavior is described in (Bengio et al., 2013) with respect to noisy rectifiers. Gradients also back-propagate through the gating network to its inputs. Our method differs here from (Bengio et al., 2015) who use boolean gates and a REINFORCE-style approach to train the gating network.

3 ADDRESSING PERFORMANCE CHALLENGES

3.1 THE SHRINKING BATCH PROBLEM

On modern CPUs and GPUs, large batch sizes are necessary for computational efficiency, so as to amortize the overhead of parameter loads and updates. If the gating network chooses k out of n experts for each example, then for a batch of b examples, each expert receives a much smaller batch of approximately $\frac{kb}{n} \ll b$ examples. This causes a naive MoE implementation to become very inefficient as the number of experts increases. The solution to this shrinking batch problem is to make the original batch size as large as possible. However, batch size tends to be limited by the memory necessary to store activations between the forwards and backwards passes. We propose the following techniques for increasing the batch size:

Mixing Data Parallelism and Model Parallelism: In a conventional distributed training setting, multiple copies of the model on different devices asynchronously process distinct batches of data, and parameters are synchronized through a set of parameter servers. In our technique, these different batches run synchronously so that they can be combined for the MoE layer. We distribute the standard layers of the model and the gating network according to conventional data-parallel schemes, but keep only one shared copy of each expert. Each expert in the MoE layer receives a combined batch consisting of the relevant examples from all of the data-parallel input batches. The same set of devices function as data-parallel replicas (for the standard layers and the gating networks) and as model-parallel shards (each hosting a subset of the experts). If the model is distributed over d devices, and each device processes a batch of size b, each expert receives a batch of approximately $\frac{kbd}{n}$ examples. Thus, we achieve a factor of d improvement in expert batch size.

In the case of a hierarchical MoE (Section B), the primary gating network employs data parallelism, and the secondary MoEs employ model parallelism. Each secondary MoE resides on one device.

2.1 门控网络

Softmax门控： 一种简单的非稀疏门控函数选择（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。

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

噪声 Top-K 门控： 我们在 Softmax 门控网络中添加了两个组件：稀疏性和噪声。在进行 soft max 函数计算之前，我们添加可调的高斯噪声，然后仅保留前 k 个值，将其余值设置为 $-\infty$（这会导致相应的门控值等于 0）。稀疏性用于节省计算量，如上所述。虽然这种形式的稀疏性会在门控函数的输出中产生一些理论上可怕的间断点，但我们尚未在实践中观察到这个问题。噪声项有助于负载均衡，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。

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

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

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

训练门控网络 我们通过简单的反向传播来训练门控网络，以及模型的其余部分。如果我们选择 $k > 1$，前 k 个专家的门控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 等人，2013) 中关于噪声整流器方面进行了描述。梯度也通过门控网络反向传播到其输入。我们的方法在这里与 (Bengio 等人，2015) 不同，后者使用布尔门控和 REINFORCE 风格的方法来训练门控网络。

3 A解决性能挑战 NGES

3.1 批量缩小问题

在现代CPU和GPU上，为了提高计算效率，需要使用较大的批量大小来摊销参数负载和更新的开销。如果门控网络为每个样本选择 k 个 n 个专家，那么对于 b 个样本的批量，每个专家接收的批量大小大约为 $\frac{kb}{n} \ll b$ 个样本。这导致在专家数量增加时，传统的MoE实现变得非常低效。解决这个批量缩小问题的方案是尽可能使原始批量大小最大化。然而，批量大小往往受限于存储正向和反向传递之间激活所需的内存。我们提出了以下技术来增加批量大小：

混合数据并行和模型并行： 在传统的分布式训练设置中，不同设备上的模型副本异步处理不同的数据批量，参数通过一组参数服务器进行同步。在我们的技术中，这些不同的批量同步运行，以便它们可以被MoE层组合。我们根据传统数据并行方案分布模型的标准层和门控网络，但只保留每个专家的一个共享副本。MoE层中的每个专家接收一个组合批量，其中包含来自所有数据并行输入批量的相关示例。相同的设备既作为标准层和门控网络的数据并行副本，也作为模型并行的分片（每个分片托管专家的子集）。如果模型分布在 d 个设备上，并且每个设备处理大小为 b 的批量，每个专家接收的批量大小大约为 $\frac{kbd}{n}$ 个样本。因此，我们在专家批量大小上实现了 d 倍的改进。

对于分层MoE（第B节），主门控网络采用数据并行，而次级MoE采用模型并行。每个次级MoE驻留在一个设备上。

This technique allows us to increase the number of experts (and hence the number of parameters) by proportionally increasing the number of devices in the training cluster. The total batch size increases, keeping the batch size per expert constant. The memory and bandwidth requirements per device also remain constant, as do the step times, as does the amount of time necessary to process a number of training examples equal to the number of parameters in the model. It is our goal to train a trillion-parameter model on a trillion-word corpus. We have not scaled our systems this far as of the writing of this paper, but it should be possible by adding more hardware.

Taking Advantage of Convolutionality: In our language models, we apply the same MoE to each time step of the previous layer. If we wait for the previous layer to finish, we can apply the MoE to all the time steps together as one big batch. Doing so increases the size of the input batch to the MoE layer by a factor of the number of unrolled time steps.

Increasing Batch Size for a Recurrent MoE: We suspect that even more powerful models may involve applying a MoE recurrently. For example, the weight matrices of a LSTM or other RNN could be replaced by a MoE. Sadly, such models break the convolutional trick from the last paragraph, since the input to the MoE at one timestep depends on the output of the MoE at the previous timestep. Gruslys et al. (2016) describe a technique for drastically reducing the number of stored activations in an unrolled RNN, at the cost of recomputing forward activations. This would allow for a large increase in batch size.

3.2 Network Bandwidth

Another major performance concern in distributed computing is network bandwidth. Since the experts are stationary (see above) and the number of gating parameters is small, most of the communication involves sending the inputs and outputs of the experts across the network. To maintain computational efficiency, the ratio of an expert's computation to the size of its input and output must exceed the ratio of computational to network capacity of the computing device. For GPUs, this may be thousands to one. In our experiments, we use experts with one hidden layer containing thousands of RELU-activated units. Since the weight matrices in the expert have sizes $input\_size \times hidden\_size$ and $hidden\_size \times output\_size$, the ratio of computation to input and output is equal to the size of the hidden layer. Conveniently, we can increase computational efficiency simply by using a larger hidden layer, or more hidden layers.

4 Balancing Expert Utilization

We have observed that the gating network tends to converge to a state where it always produces large weights for the same few experts. This imbalance is self-reinforcing, as the favored experts are trained more rapidly and thus are selected even more by the gating network. Eigen et al. (2013) describe the same phenomenon, and use a hard constraint at the beginning of training to avoid this local minimum. Bengio et al. (2015) include a soft constraint on the batch-wise average of each gate.[1]

We take a soft constraint approach. We define the importance of an expert relative to a batch of training examples to be the batchwise sum of the gate values for that expert. We define an additional loss $L_{importance}$, which is added to the overall loss function for the model. This loss is equal to the square of the coefficient of variation of the set of importance values, multiplied by a hand-tuned scaling factor $w_{importance}$. This additional loss encourages all experts to have equal importance.

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

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

[1] Bengio et al. (2015) also include two additional losses. One controls per-example sparsity, which we do not need since it is enforced by the fixed value of k. A third loss encourages diversity of gate values. In our experiments, we find that the gate values naturally diversify as the experts specialize (in a virtuous cycle), and we do not need to enforce diversity of gate values.

这种技术使我们能够通过按比例增加训练集群中的设备数量来增加专家数量（从而增加参数数量）。总批大小增加，而每个专家的批大小保持不变。每个设备的内存和带宽需求也保持不变，步长时间也保持不变，处理等于模型参数数量的训练样本所需的时间也保持不变。我们的目标是基于一个万亿词语料库训练一个万亿参数的模型。截至本文撰写时，我们尚未将系统扩展到这个程度，但通过增加更多硬件应该可以实现。

利用卷积特性：在我们的语言模型中，我们将相同的 MoE 应用于前一层每个时间步。如果我们等待前一层完成，可以将 MoE 作为一个大批次应用于所有时间步。这样做会将 MoE 层的输入批次大小增加到未展卷时间步数的倍数。

为循环 MoE 增加批次大小：我们推测更强大的模型可能涉及循环应用 MoE。例如，LSTM 或其他 RNN 的权重矩阵可以被 MoE 替换。遗憾的是，这种模型破坏了上一段中的卷积技巧，因为一个时间步的 MoE 输入依赖于前一个时间步的 MoE 输出。Gruslys 等人 (2016) 描述了一种技术，可以大幅减少展卷 RNN 中存储的激活数量，代价是重新计算前向激活。这将允许批次大小大幅增加。

3.2 网络带宽

在分布式计算中，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小，因此大部分通信涉及在网络上传输专家的输入和输出。为了保持计算效率，一个专家的计算量与其输入和输出大小的比率必须超过计算设备的计算量与其网络容量的比率。对于GPU来说，这可能是一千比一。在我们的实验中，我们使用具有一个包含数千个ReLU激活单元的隐藏层的专家。由于专家中的权重矩阵大小为 $input_size \times hidden_size$和 $hidden\_size \times output\_size$，计算量与输入和输出的比率等于隐藏层的大小。方便的是，我们只需通过使用更大的隐藏层或更多隐藏层，即可简单地提高计算效率。

4 平衡专家利用率

我们观察到门控网络倾向于收敛到一个状态，即它总是为同一少数几个专家产生较大的权重。这种不平衡是自我强化的，因为被偏爱的专家训练得更快，因此被门控网络选择得更多。Eigen等人（2013年）描述了相同的现象，并在训练开始时使用硬约束来避免这个局部最小值。Bengio等人（2015年）在每个批次上对每个门控的批平均添加了软约束.[1]

我们采用软约束方法。我们定义专家相对于一批训练样本的重要性，为该专家的批次门控值的总和。我们定义一个额外的损失 $L_{importance}$，将其添加到模型的总体损失函数中。该损失等于重要性值集合的变异系数的平方，乘以一个手工调优的缩放因子 $w_{importance}$。这个额外损失鼓励所有专家具有相等的重要性。

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

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

[1] Bengio 等人 (2015) 还包含两项额外的损失。其中一项控制每个样本的稀疏性，由于 k 的固定值已经强制实现了稀疏性，因此我们不需要它。第三项损失鼓励门值的多样性。在我们的实验中，我们发现随着专家的专业化，门值会自然多样化（形成一个良性循环），因此我们不需要强制门值的多样性。

While this loss function can ensure equal importance, experts may still receive very different numbers of examples. For example, one expert may receive a few examples with large weights, and another may receive many examples with small weights. This can cause memory and performance problems on distributed hardware. To solve this problem, we introduce a second loss function, L_{load} , which ensures balanced loads. Appendix A contains the definition of this function, along with experimental results.

5 EXPERIMENTS

5.1 1 BILLION WORD LANGUAGE MODELING BENCHMARK

Dataset: This dataset, introduced by (Chelba et al., 2013) consists of shuffled unique sentences from news articles, totaling approximately 829 million words, with a vocabulary of 793,471 words.

Previous State-of-the-Art: The best previously published results (Jozefowicz et al., 2016) use models consisting of one or more stacked Long Short-Term Memory (LSTM) layers (Hochreiter & Schmidhuber, 1997; Gers et al., 2000). The number of parameters in the LSTM layers of these models vary from 2 million to 151 million. Quality increases greatly with parameter count, as do computational costs. Results for these models form the top line of Figure 2-right.

MoE Models: Our models consist of two stacked LSTM layers with a MoE layer between them (see Figure 1). We vary the sizes of the layers and the number of experts. For full details on model architecture, training regimen, additional baselines and results, see Appendix C.

Low Computation, Varied Capacity: To investigate the effects of adding capacity, we trained a series of MoE models all with roughly equal computational costs: about 8 million multiply-and-adds per training example per timestep in the forwards pass, excluding the softmax layer. We call this metric (ops/timestep). We trained models with flat MoEs containing 4, 32, and 256 experts, and models with hierarchical MoEs containing 256, 1024, and 4096 experts. Each expert had about 1 million parameters. For all the MoE layers, 4 experts were active per input.

The results of these models are shown in Figure 2-left. The model with 4 always-active experts performed (unsurprisingly) similarly to the computationally-matched baseline models, while the largest of the models (4096 experts) achieved an impressive 24% lower perplexity on the test set.

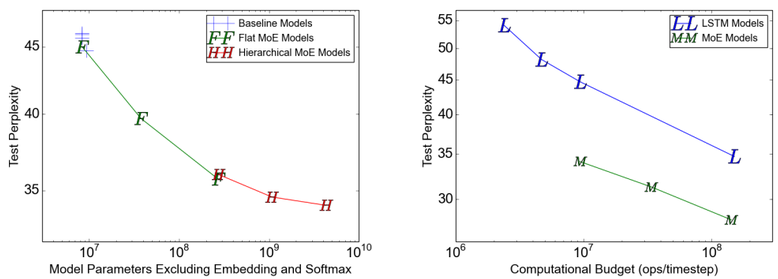

Figure 2: Model comparison on 1-Billion-Word Language-Modeling Benchmark. On the left, we plot test perplexity as a function of model capacity for models with similar computational budgets of approximately 8-million-ops-per-timestep. On the right, we plot test perplexity as a function of computational budget. The top line represents the LSTM models from (Jozefowicz et al., 2016). The bottom line represents 4-billion parameter MoE models with different computational budgets.

Varied Computation, High Capacity: In addition to the largest model from the previous section, we trained two more MoE models with similarly high capacity (4 billion parameters), but higher computation budgets. These models had larger LSTMs, and fewer but larger and experts. Details

虽然这个损失函数可以确保相等的重要性，但专家可能仍然会收到非常不同的样本数量。例如，一个专家可能收到几个权重较大的样本，而另一个专家可能收到许多权重较小的样本。这可能导致分布式硬件上的内存和性能问题。为了解决这个问题，我们引入了一个第二损失函数,L_{load} ，它确保负载均衡。附录A包含了该函数的定义，以及实验结果。

5 实验

5.1 10亿词语言建模基准

Dat集合： 该数据集由(Chelba等人，2013年)引入，包含打乱后的唯一句子 s fro m新闻文章，总计约8.29亿词，词汇量为793,471词 .

先前当前最佳： 先前发表的最佳结果(Jozefowicz等人，2016年)使用包含一个或多个堆叠的长期短期记忆(LSTM)层(（Hochreiter & Schmidhuber，1997年；Gers等人，2000年）)的模型。这些模型中LSTM层的参数数量从200万到1.51亿不等。随着参数数量的增加，质量会显著提高，计算成本也会增加。这些模型的结果构成了图2右上方的那条线。

MoE模型： 我们的模型由两层堆叠的LSTM层组成，中间夹着一个MoE层（见图1）。我们改变了层的大小和专家的数量。有关模型架构、训练方案、附加基线和结果的详细信息，请参见附录C。

低计算量，不同容量： 为了研究增加容量的影响，我们训练了一系列MoE模型，它们的计算成本大致相同：正向传递中每个训练样本约800万次乘加运算，不包括softmax层。我们称这个指标为（操作/时间步）。我们训练了包含4、32和256个专家的扁平MoE模型，以及包含256、1024和4096个专家的分层MoE模型。每个专家约有100万参数。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个始终活跃的专家的模型（毫不奇怪）表现与计算量匹配的基线模型相似，而最大的模型（4096个专家）在测试集上实现了令人印象深刻的24%的困惑度降低。

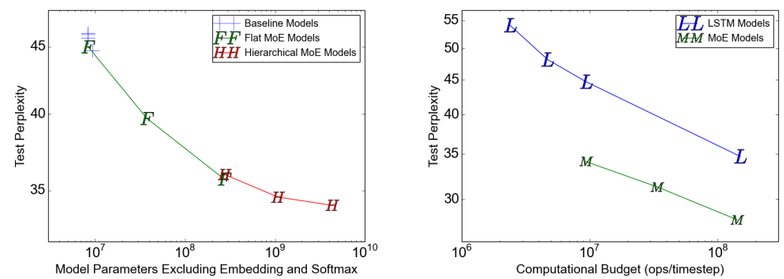

图2：在1亿词语言建模基准上的模型对比。左侧，我们绘制了测试困惑度随模型容量的变化关系，适用于计算预算相似的模型（约每时间步800万运算量）。右侧，我们绘制了测试困惑度随计算预算的变化关系。最上方的线表示来自(Jozefowicz等人，2016)的LSTM模型。最下方的线表示具有不同计算预算的4亿参数MoE模型。

多样化计算，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相似高容量的 MoE 模型（40亿参数），但计算预算更高。这些模型具有更大的 LSTM，专家数量更少但规模更大。详情

Table 1: Summary of high-capacity MoE-augmented models with varying computational budgets, vs. best previously published results (Jozefowicz et al., 2016). Details in Appendix 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**

can be found in Appendix C.2. Results of these three models form the bottom line of Figure 2-right. Table 1 compares the results of these models to the best previously-published result on this dataset . Even the fastest of these models beats the best published result (when controlling for the number of training epochs), despite requiring only 6% of the computation.

Computational Efficiency: We trained our models using TensorFlow (Abadi et al., 2016) on clusters containing 16-32 Tesla K40 GPUs. For each of our models, we determine computational efficiency in TFLOPS/GPU by dividing the number of floating point operations required to process one training batch by the observed step time and the number of GPUs in the cluster. The operation counts used here are higher than the ones we report in our ops/timestep numbers in that we include the backwards pass, we include the importance-sampling-based training of the softmax layer, and we count a multiply-and-add as two separate operations. For all of our MoE models, the floating point operations involved in the experts represent between 37% and 46% of the total.

For our baseline models wtih no MoE, observed computational efficiency ranged from 1.07-1.29 TFLOPS/GPU. For our low-computation MoE models, computation efficiency ranged from 0.74-0.90 TFLOPS/GPU, except for the 4-expert model which did not make full use of the available parallelism. Our highest-computation MoE model was more efficient at 1.56 TFLOPS/GPU, likely due to the larger matrices. These numbers represent a significant fraction of the theoretical maximum of 4.29 TFLOPS/GPU claimed by NVIDIA. Detailed results are in Appendix C, Table 7.

5.2 100 Billion Word Google News Corpus

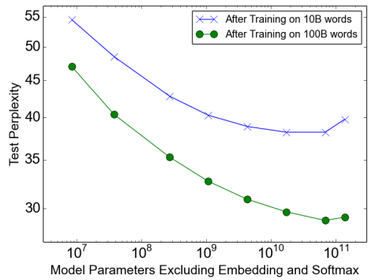

Figure 3: Language modeling on a 100 billion word corpus. Models have similar computational budgets (8 million ops/timestep).

On the 1-billion-word corpus, adding additional capacity seems to produce diminishing returns as the number of parameters in the MoE layer exceeds 1 billion, as can be seen in Figure 2-left. We hypothesized that for a larger training set, even higher capacities would produce significant quality improvements.

We constructed a similar training set consisting of shuffled unique sentences from Google's internal news corpus, totalling roughly 100 billion words. Similarly to the previous section, we tested a series of models with similar computational costs of about 8 million ops/timestep. In addition to a baseline LSTM model, we trained models augmented with MoE layers containing 32, 256, 1024,

表1：高容量MoE增强模型的摘要，计算预算不同，与Jozefowicz等人（2016年）先前发表的最佳结果对比。详情见附录C。

	Test 困惑度 10个epoch	Test 排除嵌入的困惑度 100个epoch	参数数量 和softmax层	操作/时间步	训练 Time 10个epoch	TFLOPS /GPU
最佳已发表结果	34.7	30.6	1.51亿	1.51亿	59小时，32台k40s	1.09
低成本MoE模型	34.1		4303亿	890万	15小时，32个k40s	0.74
中预算MoE模型	31.3		4313百万	3380万	17小时，64个k40s	1.22
高预算MoE模型	**28.0**		43.71亿	1.427亿	47小时，32台k40s	**1.56**

可参见附录C.2。这三个模型的结果构成了图2-右的底线。表1将这些模型的结果与该数据集上先前发表的最佳结果进行了比较。即使是最快的这些模型，在控制训练轮数的情况下也击败了先前发表的最佳结果，尽管仅需要6%的计算量。

计算效率：我们使用TensorFlow（Abadi等人，2016年）在包含16-32台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次数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ops/timestep数字中报告的数字，因为我们将反向传播包括在内，我们将softmax层的基于重要性采样的训练包括在内，我们将乘法和加法计为两个独立的操作。对于我们所有的MoE模型，专家中涉及的浮点运算占总数的37%到46%之间。

对于没有MoE的我们的基线模型，观察到的计算效率在1.07-1.29 TFLOPS/GPU之间。对于我们的低计算MoE模型，计算效率在0.74-0.90 TFLOPS/GPU之间，除了4个专家模型没有充分利用可用的并行性。我们最高计算的MoE模型效率更高，达到1.56 TFLOPS/GPU，这可能是由于更大的矩阵。这些数字代表了NVIDIA声称的理论最大值4.29 TFLOPS/GPU的显著部分。详细结果在附录C，表7中。

5.2 1000亿词谷歌新闻语料库

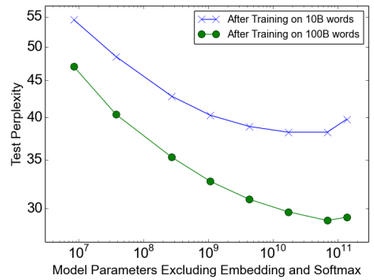

图3：在1000亿词语料库上进行语言建模。模型的计算预算相似（每时间步800万操作）。

在1亿词语料库上，随着MoE层参数数量超过1亿，增加额外容量似乎会产生边际效益递减，如图2左所示。我们假设对于更大的训练集，更高的容量将带来显著的质量提升。

我们构建了一个类似的训练集，包含从谷歌内部新闻语料库中随机排序的独特句子，总计约1000亿个词。与上一节类似，我们测试了一系列计算成本约为每时间步800万操作的模型。除了基线LSTM模型外，我们还训练了包含32、256、1024、4096、16384、65536和131072个专家的MoE层增强模型。这对应于MoE层高达1370亿个参数。架构、训练和结果的详细信息见附录D。

4096, 16384, 65536, and 131072 experts. This corresponds to up to 137 billion parameters in the MoE layer. Details on architecture, training, and results are given in Appendix D.

Results: Figure 3 shows test perplexity as a function of capacity after training on 10 billion words (top line) and 100 billion words (bottom line). When training over the full 100 billion words, test perplexity improves significantly up to 65536 experts (68 billion parameters), dropping 39% lower than the computationally matched baseline, but degrades at 131072 experts, possibly a result of too much sparsity. The widening gap between the two lines demonstrates (unsurprisingly) that increased model capacity helps more on larger training sets.

Even at 65536 experts (99.994% layer sparsity), computational efficiency for the model stays at a respectable 0.72 TFLOPS/GPU.

5.3 Machine Translation (Single Language Pair)

Model Architecture: Our model was a modified version of the GNMT model described in (Wu et al., 2016). To reduce computation, we decreased the number of LSTM layers in the encoder and decoder from 9 and 8 to 3 and 2 respectively. We inserted MoE layers in both the encoder (between layers 2 and 3) and the decoder (between layers 1 and 2). Each MoE layer contained up to 2048 experts each with about two million parameters, adding a total of about 8 billion parameters to the models. Further details on model architecture, testing procedure and results can be found in Appendix E.

Datasets: We benchmarked our method on the WMT'14 En→Fr and En→De corpora, whose training sets have 36M sentence pairs and 5M sentence pairs, respectively. The experimental protocols were also similar to those in (Wu et al., 2016): newstest2014 was used as the test set to compare against previous work (Luong et al., 2015a; Zhou et al., 2016; Wu et al., 2016), while the combination of newstest2012 and newstest2013 was used as the development set. We also tested the same model on a Google's Production English to French data.

Table 2: Results on WMT'14 En→ Fr newstest2014 (bold values represent best results).

Model	Test Perplexity	Test BLEU	ops/timenstep	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			

Table 3: Results on WMT'14 En → De newstest2014 (bold values represent best results).

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			

Table 4: Results on the Google Production En→ Fr dataset (bold values represent best results).

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

这对应于MoE层高达1370亿个参数。架构、训练和结果的详细信息见附录D。

结果： 图3显示了在训练10亿词（顶线）和100亿词（底线）后，测试困惑度随容量的变化。当在全部100亿词上训练时，测试困惑度显著提升至65536个专家（680亿参数），比计算匹配的基线降低了39%，但在131072个专家时开始下降，可能是由于稀疏性过高。两条线之间的差距扩大（不出所料地）表明，增加模型容量在更大的训练集上帮助更大。

即使在65536个专家（层稀疏性99.994%），模型的计算效率仍保持在令人满意的0.72 TFLOPS/GPU。

5.3 机器翻译（单语言对）

模型架构： 我们的模型是Wu等人（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和8减少到3和2。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）插入了MoE层。每个MoE层包含多达2048个专家，每个专家约有200万参数，为模型增加了大约80亿参数。有关模型架构、测试程序和结果的更多细节，请参见附录E。

数据集： 我们在WMT'14 En→Fr和En→De语料库上评估了我们的方法，其训练集分别有36M个句子对和5M个句子对。实验方案也与(Wu等人, 2016)中的方案相似：newstest2014被用作测试集以与先前的工作进行比较（Luong等人, 2015a; 周等人, 2016; Wu等人, 2016），而newstest2012和newstest2013的组合被用作开发集。我们还用同一模型在Google的生产型英语到法语数据上进行了测试。

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

模型	Test 困惑度	Eval BLEU	测试 ops/timenstep	总计 #参数数量	训练 Time
具有2048个专家的MoE	2.69	40.35	85M	8.7B	3天/64k40s
含2048专家的MoE（更长时间训练）	**2.63**	**40.56**	85M	8.7B	6天/64k40s
GNMT（吴等人，2016年）	2.79	39.22	214M	278M	6天/96 k80s
GNMT+RL (吴等人, 2016)	2.96	39.92	214M	278M	6天/96 k80s
PBMT (杜拉尼等人, 2014)		37.0			
LSTM (6层) (Luong等人, 2015b)		31.5			
LSTM (6层+PosUnk) (Luong等人, 2015b)		33.1			
DeepAtt (周等人, 2016)		37.7			
DeepAtt+PosUnk (周等人, 2016)		39.2			

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

模型	Test 困惑度	Test BLEU	操作/时间步	总计 参数数量	训练 Time
具有2048个专家的MoE	**4.64**	**26.03**	85M	8.7B	1天/64k40s
GNMT（吴等人，2016年）	5.25	24.91	214M	278M	1天/96k80s
GNMT +RL (吴等人, 2016)	8.08	24.66	214M	278M	1天/96k80s
PBMT (杜拉尼等人, 2014)		20.7			
DeepAtt (周等人, 2016)		20.6			

表4：在Google生产环境En→ Fr数据集上的结果（粗体值表示最佳结果 ）.

模型	Eval 困惑度	Eval BLEU	Test 困惑度	Test BLEU	操作/时间步	总计 #参数数量	训练 Time
具有2048个专家的MoE	**2.60**	**37.27**	**2.69**	**36.57**	85M	8.7B	1天/64k40s
GNMT（吴等人，2016年）	2.78	35.80	2.87	35.56	214M	278M	6天/96k80s

Results: Tables 2, 3, and 4 show the results of our largest models, compared with published results. Our approach achieved BLEU scores of 40.56 and 26.03 on the WMT'14 En→Fr and En→De benchmarks. As our models did not use RL refinement, these results constitute significant gains of 1.34 and 1.12 BLEU score on top of the strong baselines in (Wu et al., 2016). The perplexity scores are also better.[2] On the Google Production dataset, our model achieved 1.01 higher test BLEU score even after training for only one sixth of the time.

5.4 MULTILINGUAL MACHINE TRANSLATION

Dataset: (Johnson et al., 2016) train a single GNMT (Wu et al., 2016) model on a very large combined dataset of twelve language pairs. Results are somewhat worse than those for 12 separately trained single-pair GNMT models. This is not surprising, given that the twelve models have 12 times the capacity and twelve times the aggregate training of the one model. We repeat this experiment with a single MoE-augmented model. See Appendix E for details on model architecture. We train our model on the same dataset as (Johnson et al., 2016) and process the same number of training examples (about 3 billion sentence pairs). Our training time was shorter due to the lower computational budget of our model.

Results: Results for the single-pair GNMT models, the multilingual GNMT model and the multilingual MoE model are given in Table 5. The MoE model achieves 19% lower perplexity on the dev set than the multilingual GNMT model. On BLEU score, the MoE model significantly beats the multilingual GNMT model on 11 of the 12 language pairs (by as much as 5.84 points), and even beats the monolingual GNMT models on 8 of 12 language pairs. The poor performance on English → Korean seems to be a result of severe overtraining, as for the rarer language pairs a small number of real examples were highly oversampled in the training corpus.

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

	GNMT-Mono	GNMT-Multi	MoE-Multi	MoE-Multi vs. GNMT-Multi
Parameters	278M / model	278M	8.7B	
ops/timestep	212M	212M	102M	
training time, hardware	various	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

6 CONCLUSION

This work is the first to demonstrate major wins from conditional computation in deep networks. We carefully identified the design considerations and challenges of conditional computing and addressed them with a combination of algorithmic and engineering solutions. While we focused on text, conditional computation may help in other domains as well, provided sufficiently large training sets. We look forward to seeing many novel implementations and applications of conditional computation in the years to come.

ACKNOWLEDGMENTS

We would like to thank all of the members of the Google Brain and Google Translate teams who helped us with this project, in particular Zhifeng Chen, Yonghui Wu, and Melvin Johnson. Thanks also to our anonymous ICLR reviewers for the helpful suggestions on making this paper better.

[2]Reported perplexities relative to the tokenization used by both our models and GNMT.

结果：表2、3和4展示了我们最大模型的成果，并与已发表的结果进行了比较。我们的方法在WMT'14 En→Fr和En→De基准测试上取得了40.56和26.03的BLEU分数。由于我们的模型没有使用RL微调，这些结果在(Wu等人，2016)中已有的强基线之上构成了1.34和1.12的BLEU分数显著提升。困惑度分数也更好了。[2] 在Google生产数据集上，即使只训练了六分之一的时间，我们的模型测试BLEU分数也提高了1.01。

5.4 多语言机器翻译

数据集： (Johnson等人，2016年)在一个包含十二种语言对的非常大的综合数据集上训练单个GNMT（吴等人，2016年）模型。结果比12个单独训练的单语言GNMT模型稍差。考虑到十二个模型具有12倍的容量和12倍的总体训练量，这并不令人意外。我们使用单个MoE增强模型重复此实验。有关模型架构的详细信息，请参阅附录E。我们在与(Johnson等人，2016年)相同的数據集上训练我们的模型，并处理相同数量的训练样本（约30亿个句子对）。由于我们模型的计算预算较低，因此训练时间更短。

结果： 单语言GNMT模型、多语言GNMT模型和多语言MoE模型的结果如表5所示。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种语言对中的11种上显著优于多语言GNMT模型（最多高5.84分），甚至在12种语言对中的8种上优于单语言GNMT模型。在英语-韩语上的表现较差似乎是由于严重的过拟合，因为对于较少的语言对，训练语料库中的少量真实示例被严重过度采样。

表 5：多语言机器翻译（粗体值表示最佳结果）。

	GNMT-Mono	GNMT-多	MoE-多	MoE-多对多 GNMT-多
参数	278M/模型	278M	8.7B	
操作/时间步	212M	212M	102M	
训练时间，硬件	各种	21天，96k20s	**12天，64k40s**	
困惑度（开发）		4.14	**3.35**	-19%
法语 → 英语测试BLEU	36.47	34.40	**37.46**	+3.06
德语 → 英语测试BLEU	31.77	31.17	**34.80**	+3.63
日语 → 英语测试BLEU	23.41	21.62	**25.91**	+4.29
韩语 → 英语测试BLEU	25.42	22.87	**28.71**	+5.84
葡萄牙语 → 英语测试 BLEU	44.40	42.53	**46.13**	+3.60
西班牙语 → 英语测试 BLEU	38.00	36.04	**39.39**	+3.35
英语 → 法语测试 BLEU	35.37	34.00	**36.59**	+2.59
英语 → 德语测试BLEU	**26.43**	23.15	24.53	+1.38
英语 → 日语测试BLEU	**23.66**	21.10	22.78	+1.68
英语 →韩语测试BLEU	**19.75**	18.41	16.62	-1.79
英语 → 葡萄牙语测试BLEU	**38.40**	37.35	37.90	+0.55
英语 → 西班牙语测试BLEU	34.50	34.25	**36.21**	+1.96

6 结论

这项工作是首个展示深度网络中条件计算重大成果的研究。我们仔细识别了条件计算的设计考虑和挑战，并通过算法和工程解决方案加以解决。虽然我们专注于文本领域，但条件计算在其他领域也可能发挥作用，前提是拥有足够大的训练集。我们期待在未来看到更多条件计算的新颖实现和应用。

致谢

我们感谢所有帮助过我们这个项目的 Google Brain 和 Google 翻译团队的成员，特别是陈志峰、吴永辉和梅尔文·约翰逊。也感谢我们匿名的 ICLR 审稿人为改进这篇论文提出的宝贵建议。

[2]相对于我们模型和GNMT使用的分词方式，报告的困惑度。

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, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint 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, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

K. Cho and Y. Bengio. Exponentially Increasing the Capacity-to-Computation Ratio for Conditional Computation in Deep Learning. *ArXiv e-prints*, June 2014.

Ronan Collobert, Samy Bengio, and Yoshua Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computing*, 2002.

Andrew Davis and Itamar Arel. Low-rank approximations for conditional feedforward computation in deep neural networks. *arXiv preprint arXiv:1312.4461*, 2013.

Marc Peter Deisenroth and Jun Wei Ng. Distributed Gaussian processes. In *ICML*, 2015.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization, 2010.

Nadir Durrani, Barry Haddow, Philipp Koehn, and Kenneth Heafield. Edinburgh's phrase-based machine translation systems for wmt-14. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, 2014.

David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint 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.

参考文献

马丁·阿巴迪、阿什ish·阿格拉瓦尔、保罗·巴姆、尤金·布雷多、陈志峰、克雷格·西特罗、格雷戈里·S·科拉德罗、安迪·戴维斯、杰弗里·迪恩、马修·德文、桑杰伊·盖马沃特、伊恩·J·古德费洛、安德鲁·哈普、杰弗里·欧文、迈克尔·艾萨德、杨庆杰、拉法尔·约瑟夫维茨、卢卡斯·凯泽、马努贾特·库杜尔、乔什·列文伯格、丹·马内、拉贾特·蒙加、雪莉·摩尔、德雷克·戈登·默里、克里斯·奥拉、迈克·舒斯特、乔纳森·施伦斯、贝诺·斯泰纳、伊利亚·苏茨凯弗、库纳尔·塔尔瓦、保罗·A·塔克、文森特·范霍克、维杰伊·瓦苏德万、费尔南达·B·维埃加斯、奥里奥尔·维尼亚尔斯、皮特·沃登、马丁·瓦滕伯格、马丁·维克、袁宇、郑晓强。Tensorflow：在异构分布式系统上的大规模机器学习。*CoRR*, abs/1603.04467, 2016。URL http://arxiv.org/abs/1603.04467。

Rahaf Aljundi、Punarjay Chakravarty和Tinne Tuytelaars。专家门：使用专家网络的终身学习。*CoRR*，abs/1611.06194，2016年。URL http://arxiv.org/abs/1611.06194。

A. Almahairi、N. Ballas、T. Cooijmans、Y. Zheng、H.Larochelle和A.Courville。动态电容网络。*ArXiv e-prints*，2015年11月。

Dario Amodei、Rishita Anubhai、Eric Battenberg、Carl Case、Jared Casper、Bryan Catanzaro、Jing-dong 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和Zhenyao Zhu。深度语音2：英语和普通话的端到端语音识别。*arXiv preprint arXiv:1512.02595*，2015年。

Dzmitry Bahdanau、Kyunghyun Cho 和 Yoshua Bengio。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 和 Doina Precup。用于更快模型的神经网络中的条件计算。*arXiv*预印本 *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。十亿词基准用于衡量统计语言建模的进展。*arXiv*预印本 *arXiv:1312.3005*, 2013。

K. Cho 和 Y. Bengio. 深度学习中条件计算的能力-计算比率指数增长. *ArXiv e-prints*, 2014年6月.

Ronan Collobert、Samy Bengio 和 Yoshua Bengio. 用于大规模问题的并行SVM混合. 神经计算, 2002年.

Andrew Davis 和 Itamar Arel. 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年. Marc Peter Deisenroth 和 Jun Wei Ng. 分布式高斯过程. 在 *ICML* 中, 2015年. John Duchi、Elad Hazan 和 Yoram Singer. 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. 爱丁堡为wmt-14开发的基于短语的机器翻译系统. 在 第九届统计机器翻译研讨会论文集 中, 2014年.

DavidEigen、Marc'Aurelio Ranzato 和 Ilya Sutskever。在深度专家混合模型中学习因子表示。*arXiv* 预印本 *arXiv:1312.4314*，2013年。

Ekaterina Garmash 和 Christof Monz. 针对多源神经机器翻译的集成学习. 发表于 *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, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 2012.

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 backingoff 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.

Felix A. Gers, Jürgen A. Schmidhuber, 和 Fred A. Cummins. 学习遗忘：使用 LSTM 的持续预测. *Neural Computation*, 2000. Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, 和 Alex Graves. 高效的时序反向传播. *CoRR*, abs/1606.03401, 2016. URL http://arxiv.org/ abs/1606.03401. Kaiming He, Xiangyu Zhang, Shaoqing Ren, 和 Jian Sun. 用于图像识别的深度残差学习 . *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. 用于语音识别中声学建模的深度神经网络：四个研究组的共识. *IEEESignal ProcessingMagazine*, 2012. Sepp Hochreiter 和 Jürgen Schmidhuber. 长短期记忆网络. *Neural Computation*, 1997. Sergey Ioffe 和 Christian Szegedy. 批归一化：通过减少内部协变量偏移来加速深度网络训练. *arXiv preprint arXiv:1502.03167*, 2015. Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, 和 Geoffrey E. Hinton. 自适应的局部专家混合模型. *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, 和 Jeffrey Dean. Google 的多语言神经机器翻译系统：实现零样本翻译.*CoRR*, abs/1611.04558, 2016. URL http://arxiv.org/abs/1611.04558. Michael I. Jordan 和 Robert A. Jacobs. 层次专家混合模型和 EM 算法.*Neural Computing*, 1994. Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, 和 Yonghui Wu. 探索语言建模的极限. *arXiv preprint arXiv:1602.02410*, 2016. Diederik Kingma 和 Jimmy Ba. Adam：一种随机优化方法. 发表于 *ICLR*, 2015. Reinhard Kneser 和 Hermann. Ney. 改进的 m-gram 语言建模., 1995. Alex Krizhevsky, Ilya Sutskever, 和 Geoffrey E. Hinton. 使用深度卷积神经网络的 ImageNet 分类. 发表于 *NIPS*, 2012. Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeffrey Dean, 和 Andrew Y. Ng. 使用大规模无监督学习构建高级特征. 发表于 *ICML*, 2012. Patrick Gallinari Ludovic Denoyer. 深度序列神经网络. *arXivpreprintarXiv:1410.0510*, 2014. Minh-Thang Luong, Hieu Pham, 和 Christopher D. Manning. 基于注意力的神经机器翻译的有效方法. *EMNLP*, 2015a. Minh-Thang Luong, Ilya Sutskever, Quoc V. Le, Oriol Vinyals, 和 Wojciech Zaremba. 解决神经机器翻译中的罕见词问题. *ACL*, 2015b. Carl Edward Rasmussen 和 Zoubin Ghahramani. 高斯过程专家的无限混合.*NIPS*, 2002. Hasim Sak, Andrew W Senior, 和 Françoise Beaufays. 用于大规模声学建模的长短期记忆循环神经网络架构. 发表于 *INTERSPEECH*, pp. 338–342, 2014. Mike Schuster 和 Kaisuke Nakajima. 日语和韩语语音搜索. *ICASSP*, 2012. Babak Shahbaba 和 Radford Neal. 使用狄利克雷过程混合的非线性模型. *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, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint 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.

伊利亚·苏茨凯弗、奥里奥尔·维尼亚尔斯和 Quoc V. Le。神经网络的序列到序列学习。在 *NIPS*，2014 年。

Luc作为 Theis 和 Matthias Bethge。使用空间 LSTM 的生成图像建模。在 *NIPS*，2015 年 .

Volker Tresp。高斯过程混合。在 *NIPS*，2001 年。

杨辉、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：弥合人与机器翻译之间的差距。*arXiv* 预印本 *arXiv:1609.08144*，2016 年。

姚 Bangpeng，Walther Dirk，Beck Diane，以及李飞飞。 分层分类混合专家揭示了大脑区域之间的相互作用。《*NIPS*》。2009年。

Zaremba Wojciech，Sutskever Ilya，以及 Vinyals Oriol。循环神经网络正则化。《*arXivpreprint arXiv:1409.2329*》，2014年。

周杰，曹莹，王旭光，李鹏，和徐伟。用于神经机器翻译的具有快速前馈连接的深度循环模型。*arXiv* 预印本 *arXiv:1606.04199*，2016。

APPENDICES

A LOAD-BALANCING LOSS

As discussed in section 4, for load-balancing purposes, we want to define an additional loss function to encourage experts to receive roughly equal numbers of training examples. Unfortunately, the number of examples received by an expert is a discrete quantity, so it can not be used in back-propagation. Instead, we define a smooth estimator $Load(X)$ of the number of examples assigned to each expert for a batch X of inputs. The smoothness allows us to back-propagate gradients through the estimator. This is the purpose of the noise term in the gating function. We define $P(x,i)$ as the probability that $G(x)_i$ is nonzero, given a new random choice of noise on element i, but keeping the already-sampled choices of noise on the other elements. To compute $P(x,i)$, we note that the $G(x)_i$ is nonzero if and only if $H(x)_i$ is greater than the k^{th}-greatest element of $H(x)$ excluding itself. The probability works out to be:

$$P(x,i) = Pr\Big((x \cdot W_g)_i + StandardNormal() \cdot Softplus((x \cdot W_{noise})_i) \\ > kth\_excluding(H(x),k,i)\Big) \tag{8}$$

Where $kth\_excluding(v,k,i)$ means the kth highest component of v, excluding component i. Simplifying, we get:

$$P(x,i) = \Phi\Big(\frac{(x \cdot W_g)_i - kth\_excluding(H(x),k,i)}{Softplus((x \cdot W_{noise})_i)}\Big) \tag{9}$$

Where Φ is the CDF of the standard normal distribution.

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

We can now define the load loss to be the square of the coefficient of variation of the load vector, multiplied by a hand-tuned scaling factor w_{load}.

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

Initial Load Imbalance: To avoid out-of-memory errors, we need to initialize the network in a state of approximately equal expert load (since the soft constraints need some time to work). To accomplish this, we initialize the matrices W_g and W_{noise} to all zeros, which yields no signal and some noise.

Experiments: We trained a set of models with identical architecture (the MoE-256 model described in Appendix C), using different values of $w_{importance}$ and w_{load}. We trained each model for 10 epochs, then measured perplexity on the test set. We also measured the coefficients of variation in $Importance$ and $Load$, as well as ratio of the load on the most overloaded expert to the average load. This last value is significant for load balancing purposes on distributed hardware. All of these metrics were averaged over several training batches.

Table 6: Experiments with different combinations of losses.

$w_{importance}$	w_{load}	Test Perplexity	$CV(Importance(X))$	$CV(Load(X))$	$\frac{max(Load(X))}{mean(Load(X))}$
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**

附录

负载均衡损失

如第4节所述，为了负载均衡的目的，我们希望定义一个额外的损失函数来鼓励专家接收大致相等的训练样本数量。不幸的是，专家接收的样本数量是一个离散量，因此不能用于反向传播。相反，我们定义了一个平滑估计器 $Load(X)$，用于计算每个专家在输入批次 X 中被分配的样本数量。平滑性使我们能够通过估计器反向传播梯度。这是门控函数中噪声项的目的。我们定义 $P(x,i)$ 为在元素 i 上进行新的随机噪声选择，但保持其他元素上已采样的噪声选择不变的情况下，$G(x)_i$ 不为零的概率。为了计算 $P(x,i)$，我们注意到 $G(x)_i$ 不为零当且仅当 $H(x)_i$ 大于 $H(x)$ 排除自身之外的 k^{th}-最大元素。计算出的概率为：

$$P(x,i) = Pr\Big((x \cdot W_g)_i + StandardNormal() \cdot Softplus((x \cdot W_{noise})_i) \\ > kth\_excluding(H(x),k,i)\Big) \tag{8}$$

其中 $kth\_excluding(v,k,i)$ 表示 v 的第 k 个最高分量，不包括分量 i。简化后，我们得到：

$$P(x,i) = \Phi\Big(\frac{(x \cdot W_g)_i - kth\_excluding(H(x),k,i)}{Softplus((x \cdot W_{noise})_i)}\Big) \tag{9}$$

其中 Φ 是标准正态分布的 CDF。

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

现在我们可以将负载损失定义为负载向量的变异系数的平方，乘以一个手工调优的缩放因子 w_{load}。

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

初始负载不平衡：为了避免内存溢出错误，我们需要在软约束需要一些时间才能生效的状态下初始化网络，以使专家负载大致相等。为此，我们将矩阵 W_g 和 W_{noise} 初始化为零，这会产生没有信号和一些噪声。

实验：我们使用相同架构（附录C中描述的MoE-256模型）训练了一组模型，采用不同的 $w_{importance}$ 和 w_{load}值。我们为每个模型训练10个epoch，然后在测试集上测量困惑度。我们还测量了 $Importance$ 和 $Load$的变异系数，以及最过载专家的负载与平均负载的比率。这个最后值对于分布式硬件上的负载均衡非常重要。所有这些指标都在多个训练批次上进行了平均。

表6：不同损失组合的实验

$w_{importance}$	w_{load}	测试困惑度	$CV(Importance(X))$	$CV(Load(X))$	$\frac{max(Load(X))}{mean(Load(X))}$
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**

Results: Results are reported in Table 6. All the combinations containing at least one the two losses led to very similar model quality, where having no loss was much worse. Models with higher values of w_{load} had lower loads on the most overloaded expert.

B HIERACHICAL MIXTURE OF EXPERTS

If the number of experts is very large, we can reduce the branching factor by using a two-level hierarchical MoE. In a hierarchical MoE, a primary gating network chooses a sparse weighted combination of "experts", each of which is itself a secondary mixture-of-experts with its own gating network.[3] If the hierarchical MoE consists of a groups of b experts each, we denote the primary gating network by $G_{primary}$, the secondary gating networks by $(G_1, G_2..G_a)$, and the expert networks by $(E_{0,0}, E_{0,1}..E_{a,b})$. The output of the MoE is given by:

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

Our metrics of expert utilization change to the following:

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

$$Load_H(X)_{i,j} = \frac{Load_{primary}(X)_i \cdot Load_i(X^{(i)})_j}{|X^{(i)}|} \tag{14}$$

$Load_{primary}$ and $Load_i$ deonte the $Load$ functions for the primary gating network and i^{th} secondary gating network respectively. $X^{(i)}$ denotes the subset of X for which $G_{primary}(x)_i > 0$.

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 1 BILLION WORD LANGUAGE MODELING BENCHMARK - EXPERIMENTAL DETAILS

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

Model Architecture: Our model consists of five layers: a word embedding layer, a recurrent Long Short-Term Memory (LSTM) layer (Hochreiter & Schmidhuber, 1997; Gers et al., 2000), a MoE layer, a second LSTM layer, and a softmax layer. The dimensionality of the embedding layer, the number of units in each LSTM layer, and the input and output dimensionality of the MoE layer are all equal to 512. For every layer other than the softmax, we apply drouput (Zaremba et al., 2014) to the layer output, dropping each activation with probability $DropProb$, otherwise dividing by $(1 - DropProb)$. After dropout, the output of the previous layer is added to the layer output. This residual connection encourages gradient flow (He et al., 2015).

MoE Layer Architecture: Each expert in the MoE layer is a feed forward network with one ReLU-activated hidden layer of size 1024 and an output layer of size 512. Thus, each expert contains $[512 * 1024] + [1024 * 512] = 1M$ parameters. The output of the MoE layer is passed through a sigmoid function before dropout. We varied the number of experts between models, using ordinary MoE layers with 4, 32 and 256 experts and hierarchical MoE layers with 256, 1024 and 4096 experts. We call the resulting models MoE-4, MoE-32, MoE-256, MoE-256-h, MoE-1024-h and MoE-4096-h. For the hierarchical MoE layers, the first level branching factor was 16, corresponding to the number of GPUs in our cluster. We use Noisy-Top-K Gating (see Section 2.1) with $k = 4$ for the ordinary MoE layers and $k = 2$ at each level of the hierarchical MoE layers. Thus, each example is processed by exactly 4 experts for a total of 4M ops/timestep. The two LSTM layers contribute 2M ops/timestep each for the desired total of 8M.

[3] We have not found the need for deeper hierarchies.

结果： 结果报告在表6中。所有包含至少一种损失的组合都导致了非常相似的模型质量，而没有任何损失的情况则要差得多。w_{load} 值更高的模型在最过载专家上的负载更低。

B 层次专家混合

如果专家的数量非常大，我们可以通过使用一个两层的层次 MoE 来减少分支因子。在一个层次 MoE 中，一个主门控网络选择一个稀疏加权组合的“专家”，每个专家本身都是一个具有自己门控网络的二级专家混合。[3] 如果层次 MoE 由 a 组 b 专家组成，我们用 $G_{primary}$ 表示主门控网络，用 $(G_1, G_2..G_a)$ 表示二级门控网络，用 $(E_{0,0}, E_{0,1}..E_{a,b})$ 表示专家网络。MoE 的输出为：

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

我们的专家利用率指标变为以下：

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

$$Load_H(X)_{i,j} = \frac{Load_{primary}(X)_i \cdot Load_i(X^{(i)})_j}{|X^{(i)}|} \tag{14}$$

$Load_{primary}$ 和 $Load_i$ 分别表示主门控网络和 $Load$ 次级门控网络的 i^{th} 函数。$X^{(i)}$ 表示 X 的子集，对于该子集 $G_{primary}(x)_i > 0$。

似乎让 $Load_H(X)_{i,\ j} = Load_i(X_i)_j$ 会更简单，但这不会对主要的门控网络有梯度，所以我们使用上面的公式。

C 1 十亿词语言建模基准 - 实验细节

C.1 每步 800 万次运算的模型

模型架构：我们的模型由五层组成：一个词嵌入层、一个循环长短期记忆（LSTM）层（Hochreiter & Schmidhuber, 1997; Gers 等人, 2000）、一个 MoE 层、一个第二 LSTM 层和一个 softmax 层。嵌入层的维度、每个 LSTM 层中的单元数以及 MoE 层的输入和输出维度都等于 512。对于 softmax 层以外的每一层，我们对层输出应用 dropout（Zaremba 等人, 2014），以概率 $DropProb$ 每次丢弃激活，否则除以 $(1 - DropProb)$。dropout 后，将前一层的输出加到层输出上。这种残差连接鼓励梯度流（He 等人, 2015）。

MoE层架构： MoE层中的每个专家是一个前馈网络，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$[512 * 1024] + [1024 * 512] = 1M$ 个参数。MoE层的输出在dropout之前通过sigmoid函数处理。我们在模型之间改变了专家的数量，使用具有4、32和256个专家的普通MoE层，以及具有256、1024和4096个专家的分层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门控（见第2.1节）为普通MoE层使用 $k = 4$，并在分层MoE层的每个级别使用 $k = 2$。因此，每个示例由恰好4个专家处理，总计4M ops/timestep。这两个LSTM层各自贡献2M ops/timestep，以达到所需的8M总量。

[3] 我们没有发现需要更深的层次结构。

Computationally-Matched Baselines: The MoE-4 model does not employ sparsity, since all 4 experts are always used. In addition, we trained four more computationally-matched baseline models with no sparsity:

- MoE-1-Wide: The MoE layer consists of a single "expert" containing one ReLU-activated hidden layer of size 4096.
- MoE-1-Deep: The MoE layer consists of a single "expert" containing four ReLU-activated hidden layers, each with size 1024.
- 4xLSTM-512: We replace the MoE layer with two additional 512-unit LSTM layers.
- LSTM-2048-512: The model contains one 2048-unit LSTM layer (and no MoE). The output of the LSTM is projected down to 512 dimensions (Sak et al., 2014). The next timestep of the LSTM receives the projected output. This is identical to one of the models published in (Jozefowicz et al., 2016). We re-ran it to account for differences in training regimen, and obtained results very similar to the published ones.

Training: The models were trained on a cluster of 16 K40 GPUs using the synchronous method described in Section 3. Each batch consisted of a set of sentences totaling roughly 300,000 words. In the interest of time, we limited training to 10 epochs, (27,000 steps). Training took 12-16 hours for all models, except for MoE-4, which took 18 hours (since all the expert computation was performed on only 4 of 16 GPUs). We used the Adam optimizer (Kingma & Ba, 2015). The base learning rate was increased linearly for the first 1000 training steps, and decreased after that so as to be proportional to the inverse square root of the step number. The Softmax output layer was trained efficiently using importance sampling similarly to the models in (Jozefowicz et al., 2016). For each model, we performed a hyper-parmeter search to find the best dropout probability, in increments of 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.

Results: We evaluate our model using perplexity on the holdout dataset, used by (Chelba et al., 2013; Jozefowicz et al., 2016). We follow the standard procedure and sum over all the words including the end of sentence symbol. Results are reported in Table 7. For each model, we report the test perplexity, the computational budget, the parameter counts, the value of $DropProb$, and the computational efficiency.

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**

计算匹配基线： MoE-4模型不采用稀疏性，因为所有4个专家始终被使用。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层包含一个包含一个4096大小的ReLU激活隐藏层的“专家”。
- MoE-1-Deep: MoE层包含一个包含四个1024大小ReLU激活隐藏层的“专家”。
- 4xLSTM-512: 我们用两个额外的512单元LSTM层替换MoE层。
- LSTM-2048-512: 该模型包含一个2048单元LSTM层（且没有MoE）。LSTM的输出被投影到512维（Sak等人，2014年）。LSTM的下一个时间步接收投影后的输出。这与(Jozefowicz等人，2016年)发表的一个模型相同。我们重新运行它以考虑训练方案的不同，并获得了与已发表结果非常相似的结果。

训练： 模型在由16个K40 GPU组成的集群上使用第3节中描述的同步方法进行训练。每个批次包含一组句子，总字数约为30万。为了节省时间，我们将训练限制在10个epoch（27,000步）。除MoE-4模型外（该模型所有专家计算仅在16个GPU中的4个上进行，因此耗时18小时），所有模型的训练时间均为12-16小时。我们使用了Adam优化器（Kingma & Ba, 2015）。基础学习率在前1000个训练步长内线性增加，之后按步长数的平方根倒数比例降低。Softmax输出层使用与(Jozefowicz等人，2016)中的模型类似的重要性采样方法高效训练。对于每个模型，我们通过以0.1为增量进行超参数搜索，以找到最佳dropout概率。

为确保专家利用均衡，我们设置了 $w_{importance} = 0.1$ 和 $w_{load} = 0.1$，如第4节和附录A所述。

结果： 我们使用困惑度在留出集上评估我们的模型，该留出集由 (Chelba 等人，2013; Jozefowicz 等人，2016) 使用。我们遵循标准程序，并对所有单词（包括句子结束符号）求和。结果报告在表7中。对于每个模型，我们报告测试困惑度、计算预算、参数数量、$DropProb$的值以及计算效率。

表7：在1亿词语言建模基准上的模型对比。带*的模型来自(Jozefowicz等人，2016年)。

模型	Test 困惑度 10个epoch	Test 困惑度 (最终)	操作/时间步 (百万)	参数数量（不包括）嵌入 &softmax (百万)	总计 #参数 (十亿)	$Drop-Prob$	TFLOPS 每张GPU (观察到)
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 More Expensive Models

We ran two additional models (MoE-34M and MoE-143M) to investigate the effects of adding more computation in the presence of a large MoE layer. These models have computation budgets of 34M and 143M ops/timestep. Similar to the models above, these models use a MoE layer between two LSTM layers. The dimensionality of the embedding layer, and the input and output dimensionality of the MoE layer are set to 1024 instead of 512. For MoE-34M, the LSTM layers have 1024 units. For MoE-143M, the LSTM layers have 4096 units and an output projection of size 1024 (Sak et al., 2014). MoE-34M uses a hierarchical MoE layer with 1024 experts, each with a hidden layer of size 2048. MoE-143M uses a hierarchical MoE layer with 256 experts, each with a hidden layer of size 8192. Both models have 4B parameters in the MoE layers. We searched for the best $DropProb$ for each model, and trained each model for 10 epochs.

The two models achieved test perplexity of 31.3 and 28.0 respectively, showing that even in the presence of a large MoE, more computation is still useful. Results are reported at the bottom of Table 7. The larger of the two models has a similar computational budget to the best published model from the literature, and training times are similar. Comparing after 10 epochs, our model has a lower test perplexity by 18%.

D 100 Billion Word Google News Corpus - Experimental Details

Model Architecture: The models are similar in structure to the 8-million-operations-per-timestep models described in the previous section. We vary the number of experts between models, using an ordinary MoE layer with 32 experts and hierarchical MoE layers with 256, 1024, 4096, 16384, 65536 and 131072 experts. For the hierarchical MoE layers, the first level branching factors are 32, 32, 64, 128, 256 and 256, respectively.

Training: Models are trained on a cluster of 32 Tesla K40 GPUs, except for the last two models, which are trained on clusters of 64 and 128 GPUs so as to have enough memory for all the parameters. For all models, training batch sizes are approximately 2.5 million words. Models are trained once-through over about 100 billion words.

We implement several memory optimizations in order to fit up to 1 billion parameters per GPU. First, we do not store the activations of the hidden layers of the experts, but instead recompute them on the backwards pass. Secondly, we modify the optimizer on the expert parameters to require less auxiliary storage:

The Adam optimizer (Kingma & Ba, 2015) keeps first and second moment estimates of the per-parameter gradients. This triples the required memory. To avoid keeping a first-moment estimator, we set $\beta_1 = 0$. To reduce the size of the second moment estimator, we replace it with a factored approximation. For a matrix of parameters, instead of maintaining a full matrix of second-moment estimators, we maintain vectors of row-wise and column-wise averages of that matrix. At each step, the matrix of estimators is taken to be the outer product of those two vectors divided by the mean of either one. This technique could similarly be applied to Adagrad (Duchi et al., 2010).

Table 8: Model comparison on 100 Billion Word Google News Dataset

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

Results: We evaluate our model using perplexity on a holdout dataset. Results are reported in Table 8. Perplexity after 100 billion training words is 39% lower for the 68-billion-parameter MoE

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 层的情况下增加计算量的影响。这些模型的计算预算为 34M 和 143M ops/timestep。与上面的模型类似，这些模型在两个 LSTM 层之间使用一个 MoE 层。嵌入层的维度以及 MoE 层的输入和输出维度设置为 1024 而不是 512。对于 MoE-34M，LSTM 层有 1024 个单元。对于 MoE-143M，LSTM 层有 4096 个单元和一个大小为 1024 的输出投影（Sak 等人，2014 年）。MoE-34M 使用一个具有 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 使用一个具有 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。这两个模型在 MoE 层中都有 4B 个参数。我们为每个模型搜索了最佳 $DropProb$，并训练每个模型 10 个 epoch。

这两个模型分别达到了31.3和28.0的测试困惑度，表明即使存在大型专家混合层（MoE），更多的计算仍然是有用的。结果报告在表7的底部。这两个模型中较大的那个，其计算预算与文献中已发表的最好模型相似，训练时间也相似。比较10个epoch后的结果，我们的模型测试困惑度降低了18%。

D 1000亿词谷歌新闻语料库 - 实验细节

模型架构：这些模型在结构上与上一节中描述的每一步操作量为800万的模型相似。我们通过改变专家数量来调整模型，使用一个包含32个专家的普通专家混合层（MoE层），以及包含256、1024、4096、16384、65536和131072个专家的分层专家混合层。对于分层专家混合层，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组成的集群上进行训练，但最后两个模型则分别使用64个和128个GPU组成的集群进行训练，以便为所有参数提供足够的内存。对于所有模型，训练批次大小约为250万词。模型一次性在约1000亿词上进行训练。

我们实现了几种内存优化，以便每张GPU上最多能容纳10亿个参数。首先，我们不存储专家隐藏层的激活值，而是在反向传播时重新计算它们。其次，我们修改了专家参数的优化器，以减少辅助存储需求：

Adam优化器（Kingma & Ba，2015）保持每个参数梯度的第一和第二矩估计。这使所需的内存增加了三倍。为了避免保持一个第一矩估计器，我们设置 $\beta_1 = 0$。为了减小第二矩估计器的大小，我们用分解近似来替换它。对于一个参数矩阵，我们不是维护一个完整的第二矩估计器矩阵，而是维护该矩阵的行向和列向平均值向量。在每一步，估计器矩阵被取为这两个向量的外积除以其中一个向量的均值。这种技术同样可以应用于Adagrad（Duchi等人，2010）。

表8：在1000亿词谷歌新闻数据集上的模型对比

模型	Test 困惑度 .1个epoch	Test 困惑度 1个epoch	操作/时间步 (百万)	#参数（不包括）嵌入. &softmax (百万)	总计 #参数 (数十亿)	TFLOPS 每张GPU (观测)
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：我们使用困惑度在留出集上评估我们的模型。结果报告在表8中。在训练1000亿个词后，68亿参数的MoE模型的困惑度比基线模型降低了39%

model than for the baseline model. It is notable that the measured computational efficiency of the largest model (0.30 TFLOPS/GPU) is very low compared to the other models. This is likely a result of the fact that, for purposes of comparison to the other models, we did not increase the training batch size proportionally to the number of GPUs. For comparison, we include results for a computationally matched baseline model consisting of 4 LSTMs, and for an unpruned 5-gram model with Kneser-Ney smoothing (Kneser & Ney, 1995).[4]

E MACHINE TRANSLATION - EXPERIMENTAL DETAILS

Model Architecture for Single Language Pair MoE Models: Our model is a modified version of the GNMT model described in (Wu et al., 2016). To reduce computation, we decrease the number of LSTM layers in the encoder and decoder from 9 and 8 to 3 and 2 respectively. We insert MoE layers in both the encoder (between layers 2 and 3) and the decoder (between layers 1 and 2). We use an attention mechanism between the encoder and decoder, with the first decoder LSTM receiving output from and providing input for the attention [5]. All of the layers in our model have input and output dimensionality of 512. Our LSTM layers have 2048 hidden units, with a 512-dimensional output projection. We add residual connections around all LSTM and MoE layers to encourage gradient flow (He et al., 2015). Similar to GNMT, to effectively deal with rare words, we used sub-word units (also known as "wordpieces") (Schuster & Nakajima, 2012) for inputs and outputs in our system.

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).

We train models with different numbers of experts in the MoE layers. In addition to a baseline model with no MoE layers, we train models with flat MoE layers containing 32 experts, and models with hierarchical MoE layers containing 512 and 2048 experts. The flat MoE layers use $k = 4$ and the hierarchical MoE models use $k = 2$ at each level of the gating network. Thus, each input is processed by exactly 4 experts in each MoE layer. Each expert in the MoE layer is a feed forward network with one hidden layer of size 2048 and ReLU activation. Thus, each expert contains $[512 * 2048] + [2048 * 512] = 2M$ parameters. The output of the MoE layer is passed through a sigmoid function. We use the strictly-balanced gating function described in Appendix F.

Model Architecture for Multilingual MoE Model: We used the same model architecture as for the single-language-pair models, with the following exceptions: We used noisy-top-k gating as described in Section 2.1, not the scheme from Appendix F. The MoE layers in the encoder and decoder are non-hierarchical MoEs with $n = 512$ experts, and $k = 2$. Each expert has a larger hidden layer of size 8192. This doubles the amount of computation in the MoE layers, raising the computational budget of the entire model from 85M to 102M ops/timestep.

Training: We trained our networks using the Adam optimizer (Kingma & Ba, 2015). The base learning rate was increased linearly for the first 2000 training steps, held constant for an additional 8000 steps, and decreased after that so as to be proportional to the inverse square root of the step number. For the single-language-pair models, similarly to (Wu et al., 2016), we applied dropout (Zaremba et al., 2014) to the output of all embedding, LSTM and MoE layers, using $DropProb = 0.4$. Training was done synchronously on a cluster of up to 64 GPUs as described in section 3. Each training batch consisted of a set of sentence pairs containing roughly 16000 words per GPU.

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]While the original size of the corpus was 130 billion words, the neural models were trained for a maximum of 100 billion words. The reported Kneser-Ney 5-gram models were trained over 13 billion and 130 billion words respectively, giving them a slight advantage over the other reported results.

[5]For performance reasons, we use a slightly different attention function from the one described in (Wu et al., 2016) - See Appendix G

模型更低。值得注意的是，最大模型的测量计算效率（0.30 TFLOPS/GPU）与其他模型相比非常低。这很可能是由于，为了与其他模型进行比较，我们没有将训练批大小与GPU数量成比例增加。为了比较，我们包括了由4个LSTM组成的计算上匹配的基线模型的结果，以及一个未剪枝的5-gram模型（Kneser & Ney, 1995）的Kneser-Ney平滑结果。[4]

E 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等人(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和8减少到3和2。我们在编码器（在第2层和第3层之间）和解码器（在第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接收来自注意力 [5]的输出并提供输入。我们模型中的所有层都具有512的输入和输出维度。我们的LSTM层有2048个隐藏单元，带有512维度的输出投影。我们围绕所有LSTM和MoE层添加了残差连接，以鼓励梯度流（He等人，2015）。与GNMT类似，为了有效处理罕见词，我们使用子词单元（也称为"词片"）(Schuster & Nakajima, 2012)作为我们的系统输入和输出。

We 使用32K词素的共享源和目标词汇表。我们还使用了(Wu等人, 2016)中提出的技术。sear使用门控网络。

我们在MoE层中训练具有不同专家数量的模型。除了没有MoE层的基线模型外，我们还训练了包含32个专家的扁平MoE层模型，以及包含512和2048个专家的分层MoE层模型。扁平MoE层使用 $k = 4$，分层MoE模型在每个门控网络级别使用 $k = 2$。因此，每个MoE层中的每个输入都由恰好4个专家处理。MoE层中的每个专家是一个具有2048大小隐藏层和ReLU激活的前馈网络。因此，每个专家包含 $[512 * 2048] + [2048 * 512] = 2M$ 个参数。MoE层的输出通过一个Sigmoid函数。我们使用附录F中描述的严格平衡门控函数。

多语言MoE模型的结构：我们使用了与单语言对模型相同的模型结构，但有以下例外：我们使用了第2.1节中描述的噪声top-k门控，而不是附录F中的方案。编码器和解码器中的MoE层是非层次化的MoEs，具有 $n = 512$ 个专家，和 $k = 2$。每个专家都有一个更大的隐藏层，大小为8192。这使MoE层的计算量翻倍，将整个模型的计算预算从8500万增加到102M操作/时间步。

训练：我们使用Adam优化器（Kingma & Ba, 2015）训练了我们的网络。基础学习率在前2000个训练步骤内线性增加，然后在额外的8000个步骤内保持不变，之后按步骤数的平方根倒数比例下降。对于单语言对模型，类似于（吴等人，2016），我们将dropout（Zaremba et al., 2014）应用于所有嵌入、LSTM和MoE层的输出，使用 $DropProb$ =0。4。训练是在最多64个GPU组成的集群上同步进行的，如第3节所述。每个训练批次包含一组句子对，每个GPU包含大约16000个单词。

为确保专家利用均衡，我们设置了 $w_{importance} = 0.01$ 和 $w_{load} = 0.01$，如第 4 节和附录 A 所述。

指标：我们使用困惑度和标准BLEU分数指标评估了我们的模型。我们报告了由multi-bleu.pl脚本计算的分词BLEU分数，该脚本从Moses的公共实现（在Github上）下载，该实现也用于(Luong等人，2015a)。

[4]虽然语料库的原始大小为 1300 亿词，但神经模型最多训练 1000 亿词。报告的 Kneser-Ney 5-gram 模型分别基于 130 亿和 1300 亿词进行训练，因此比其他报告结果略有优势。[5]出于性能原因，我们使用了与 (吴等人, 2016) 中描述略有不同的注意力函数——参见附录 G

Results: Tables 2, 3 and 4 in Section 5.3 show comparisons of our results to other published methods. Figure 4 shows test perplexity as a function of number of words in the (training data's) source sentences processed for models with different numbers of experts. As can be seen from the Figure, as we increased the number of experts to approach 2048, the test perplexity of our model continued to improve.

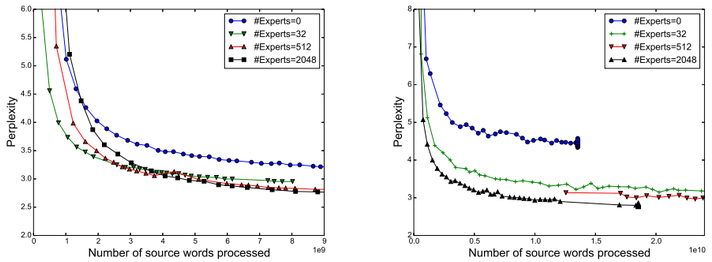

Figure 4: Perplexity on WMT'14 En→ Fr (left) and Google Production En→ Fr (right) datasets as a function of number of words processed. The large differences between models at the beginning of training are due to different batch sizes. All models incur the same computational budget (85M ops/timestep) except the one with no experts.

We found that the experts indeed become highly specialized by syntax and/or semantics, as can be seen in Table 9. For example, one expert is used when the indefinite article "a" introduces the direct object in a verb phrase indicating importance or leadership.

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 ...
...	...	...

F STRICTLY BALANCED GATING

Due to some peculiarities in our infrastructure which have since been fixed, at the time we ran some of the machine translation experiments, our models ran faster if every expert received exactly the same batch size. To accommodate this, we used a different gating function which we describe below.

Recall that we define the softmax gating function to be:

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

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:

结果： 第5.3节的表2、3和4展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（训练数据源句）中单词数量的函数，对于具有不同专家数量的模型。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持续改进。

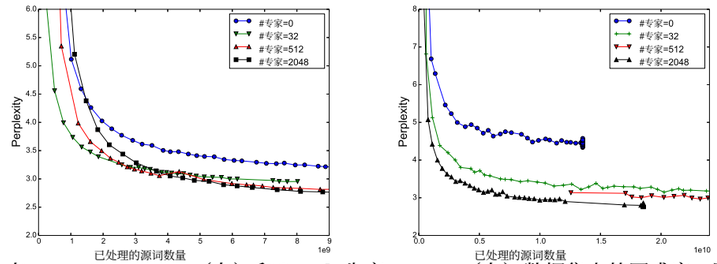

图4：在WMT'14 En→ Fr（左）和Google生产En→ Fr（右）数据集上的困惑度，随处理单词数量的变化。训练初期模型之间的巨大差异是由于不同的批处理大小。除没有专家的模型外，所有模型都使用相同的计算预算（8500万操作/时间步）。

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"在动词短语中引入直接宾语并表明重要性或领导力时，会使用一个专家。

表9：WMT'14 En→ Fr翻译模型编码器部分MoE层中2048个专家中几个专家的上下文。对于每个专家 i，我们将训练批次中的输入按 $G(x)_i$的降序排序，并显示输入句子中相应位置的周围单词。

专家381	专家752	专家2004
...与**研究人员**，...	...扮演**核心**...	...与**快速**增长的...
...致力于**创新**。	...扮演**关键**...	...在**静态**条件下...
...指标**研究人员**。	...提供<样式 id='1'>立法</样式>...	...以<样式 id='1'>迅速</样式>地...
...<样式 id='1'>生成</样式>...	...发挥<样式 id='1'>主导</样式>作用...	...以<样式 id='1'>极端</样式>地...
...技术<样式 id='1'>创新</样式>是...	...假设<样式 id='1'>领导</样式>...	...<样式 id='1'>迅速</样式>的...
...技术<样式 id='1'>创新</样式>，...	...起<样式 id='1'>核心</样式>作用...	...最<样式 id='1'>快</样式>的...
...通过<样式 id='1'>创新</样式>支持...	... taken一 领先 ...	... the**快速** 方法 ...
...角色 **创新** 将 ...	...建立一 和解 ...	... rec **当前**) ...
...研究**科学** 家 ...	...扮演了一个至关重要的...	...提供**快速**访问...
...促进**创新**...	...拥有一**个**核心的...	...含有**易挥发的**有机...
...	...	...

F 严格平衡门控

由于我们基础设施中存在一些特殊问题，这些问题现已解决，在运行部分机器翻译实验时，如果每个专家都接收完全相同的批次大小，我们的模型运行速度会更快。为了适应这一点，我们使用了不同的门控函数，具体描述如下。

回想一下，我们定义softmax门控函数为：

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

稀疏门控（另一种表述方式）： 为了获得稀疏门控向量，我们将 $G_\sigma(x)$ 与稀疏掩码 $M(G_\sigma(x))$ 按元素相乘，并对输出进行归一化。该掩码本身是 $G_\sigma(x)$ 的函数，用于指定哪些专家被分配给每个输入示例：

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

Top-K Mask: To implement top-k gating in this formulation, we would let $M(v) = TopK(v, k)$, where:

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

Batchwise Mask: To force each expert to receive the exact same number of examples, we introduce an alternative mask function, $M_{batchwise}(X, m)$, which operates over batches of input vectors. Instead of keeping the top k values per example, we keep the top m values per expert across the training batch, where $m = \frac{k|X|}{n}$, so that each example is sent to an average of k experts.

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

As our experiments suggest and also observed in (Ioffe & Szegedy, 2015), using a batchwise function during training (such as $M_{batchwise}$) requires modifications to the inference when we may not have a large batch of examples. Our solution to this is to train a vector T of per-expert threshold values to approximate the effects of the batchwise mask. We use the following mask at inference time:

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

To learn the threshold values, we apply an additional loss at training time which is minimized when the batchwise mask and the threshold mask are identical.

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

G ATTENTION FUNCTION

The attention mechanism described in GNMT (Wu et al., 2016) involves a learned "Attention Function" $A(x_i, y_j)$ which takes a "source vector" x_i and a "target vector" y_j, and must be computed for every source time step i and target time step j. In GNMT, the attention function is implemented as a feed forward neural network with a hidden layer of size n. It can be expressed as:

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

Where U and W are trainable weight matrices and V is a trainable weight vector.

For performance reasons, in our models, we used a slightly different attention function:

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

With our attention function, we can simultaneously compute the attention function on multiple source time steps and multiple target time steps using optimized matrix multiplications. We found little difference in quality between the two functions.

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

Top-K 掩码：为了在这个表述方式中实现 Top-K 门控，我们将让 $M(v) = TopK(v, k)$，其中：

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

批处理掩码：为了迫使每个专家接收完全相同数量的示例，我们引入了一种替代的掩码函数，$M_{batchwise}(X, m)$，它作用于输入向量的批次上。我们不是保留每个示例的顶部 k 值，而是保留每个专家在训练批次中的顶部 m 值，其中 $m = \frac{k|X|}{n}$，以便每个示例被发送到平均 k 个专家。

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

正如我们的实验所暗示的，并且在(Ioffe & Szegedy, 2015)中观察到的那样，在训练期间使用批处理函数（例如 $M_{batchwise}$）需要在可能没有大量示例时对推理进行修改。我们对此的解决方案是训练一个向量 T，其中包含每个专家的阈值，以近似批处理掩码的效果。我们在推理时使用以下掩码：

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

为了学习阈值，我们在训练时应用了一个附加损失，当批处理掩码和阈值掩码相同时，该损失会被最小化。

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

G 注意力函数

GNMT（吴等人，2016）中描述的注意力机制涉及一个学习的"注意力函数" $A(x_i, y_j)$，该函数接收一个"源向量" x_i 和一个"目标向量" y_j，并且必须为每个源时间步 i 和目标时间步 j进行计算。在GNMT中，注意力函数被实现为一个具有大小为 n的隐藏层的前馈神经网络。它可以表示为：

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

其中 U 和 W 是可训练的权重矩阵，V 是一个可训练的权重向量。

出于性能原因，在我们的模型中，我们使用了一个略有不同的注意力函数：

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

使用我们的注意力函数，我们可以同时通过优化的矩阵乘法在多个源时间步和多个目标时间步上计算注意力函数。我们发现这两种函数在质量上几乎没有差异。