

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

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

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

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

ABSTRACT

The capacity of a neural network to absorb information is limited by its number of parameters. In this work, we present a new kind of layer, the Sparsely-Gated Mixture-of-Experts (MoE), which can be used to effectively increase model capacity with only a modest increase in computation. This layer consists of up to thousands of feed-forward sub-networks (experts) containing a total of up to billions of parameters. A trainable gating network determines a sparse combination of these experts to use for each example. We apply the MoE to the task of language modeling, where model capacity is critical for absorbing the vast quantities of world knowledge available in the training corpora. We present new language model architectures where an MoE layer is inserted between stacked LSTMs, resulting in models with orders of magnitude more parameters than would otherwise be feasible. On language modeling and machine translation benchmarks, we achieve comparable or better results than state-of-the-art at lower computational cost, including test perplexity of 28.0 on the 1 Billion Word Language Modeling Benchmark and BLEU scores of 40.56 and 26.03 on the WMT'14 En to Fr and En to De datasets respectively.

1 INTRODUCTION

The exponential growth in training data as well as the increasing size and complexity of neural networks have been persistent trends in machine learning. It is evident that realizing the true potential of larger datasets demands training models with larger numbers of parameters. 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. To this end, it is imperative to design new scalable techniques that enable training ever larger models on ever larger datasets.

In this work we present a new technique that enables training extremely large models without increasing computation. Our approach is to increase the number of parameters by introducing a new type of neural network layer: a Sparsely-Gated Mixture-of-Experts (MoE). The MoE layer consists of a number of experts, each being a neural network model, and a trainable gating network which selects a sparse combination of the experts to process each input (see Figure 1). The sparse gating allows us to make the layer extremely large (billions of parameters), while keeping computation per example manageable (millions of FLOPs). We describe techniques for keeping the experts' batch sizes large enough not to sacrifice efficient GPU matrix multiplication even when using 99.9% sparsity.

While the concept of a mixture of experts has existed for some time, e.g., (Jacobs et al., 1991), our work advances prior art in several key aspects: We introduce the sparsity in the gating network which allows us to increase the number of parameters while keeping the computation constant. We

^{*}Equally major contributors

[†]Work done as a member of the Google Brain Residency program (g.co/brainresidency)

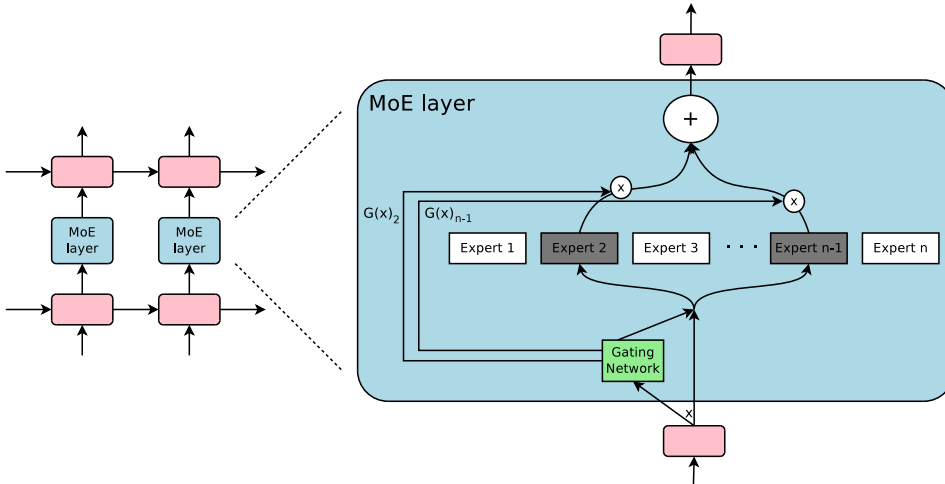


Figure 1: A Mixture of Experts (MoE) layer embedded within a language model. In this case, the sparse gating function selects two experts to perform computations. Their outputs are modulated by the outputs of the gating network.

also pose MoEs as a general purpose neural network component which can be applied in a feed-forward, recurrent or convolutional manner. In addition, we develop novel system design practices that enable scalable and parallel training of MoE layers with thousands of experts.

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 as in Figure 1. The MoE is called once for each position in the text, selecting a different combination of experts at each position. The different experts tend to become highly specialized based on syntax and semantics. On both language modeling and machine translation benchmarks, we improve on best published results at a fraction of the computational cost.

2 RELATED WORK

Exploiting scale in both the number of available training examples and the model sizes has been central to the success of deep learning. When datasets are sufficiently large, increasing the size 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). Thus, new mechanisms should be designed to enable scalable training of very large models on large datasets.

Ever 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. Prior work has focused on different aspects including using different types of expert models such as SVMs (Collobert et al., 2002), Gaussian Processes (Tresp, 2001; Theis & Bethge, 2015; Deisenroth & Ng, 2015), Dirichlet Processes (Shahbaba & Neal, 2009), or different expert configurations such as a hierarchical structure (Yao et al., 2009), and infinite number of experts (Rasmussen & Ghahramani, 2002). Eigen et al. (2013) extends MoE to a deep model by stacking two layers of mixture of experts (with each expert being a feed forward network) followed by a Softmax layer. The experts output are weighted by a trainable gating network. Garmash & Monz (2016) suggests 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 key difference between our work and prior research on MoE is that our work enables training extremely large number of experts (e.g., thousands) and parameters (e.g., multiple billions). This is made possible by a trainable gating network that assigns the inputs to only a sparse number of experts. We regard our MoE as a new type of neural network layer that can be used to increase the capacity of any conventional deep learning model in a computationally tractable manner. Our

experts and the gating function are learnable feed forward networks. In our experiments, we create models with billions of parameters whose training time is comparable or even better than that of previous state-of-the-art models that have only millions of parameters. We show that such dramatic increase in parameter size effectively results in new state-of-the-art inference accuracy in language modeling and translation tasks.

3 THE STRUCTURE OF THE MIXTURE-OF-EXPERTS LAYER

The Mixture-of-Experts (MoE) layer consists of a set of n “expert networks” $E_1, \dots, E_n$, 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:

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

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.

3.1 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.

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

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. By selecting more than one expert, we obtain backpropagated derivatives for the probabilities of the selected experts and these derivatives train the gating network to increase the probabilities associated with the more useful of the selected experts. The noise term helps with load balancing and will be discussed below. The amount of noise per component is controlled by a second trainable weight matrix W_{noise} .

$$G(x) = \text{Softmax}(\text{KeepTopK}(x \cdot W_g + \text{Noise}(x), k)) \quad (3)$$

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

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

The parameter k , which

3.2 ENSURING EQUAL 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. To encourage the experts to be used equally across each training batch, we impose a soft constraint in the form of an additional loss function which is added to the overall loss function for the model. For a batch of inputs X , The loss function is computed as:

$$L(X) = w_{importance} \cdot CV(Importance(X))^2 + w_{load} \cdot CV(Load(X))^2 \quad (6)$$

$Importance(X)$ and $Load(X)$ are n -dimensional vectors measuring per-expert utilization, which we will describe below. CV denotes the coefficient-of-variation of the components of a vector.¹ $w_{importance}$ and w_{load} are hand-tuned scalar weights.

Importance: The simplest measure of per-expert utilization is to sum the output of the gating network across the batch of examples:

$$Importance(X) = \sum_{x \in X} G(x) \quad (7)$$

Load: When running the expert networks on different devices, it is imperative for time and memory purposes that the different experts process similar numbers of examples. 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 was 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. This works out to be:

$$P(x, i) = \Phi\left(\frac{(x \cdot W_g)_i - kth_excluding(x \cdot W_g + Noise(x), k, i)}{Softplus((x \cdot W_{noise})_i)}\right) \quad (8)$$

Where Φ is the CDF of the standard normal distribution, and $kth_excluding(v, k, i)$ means the k th highest component of v , excluding component i .

$$Load(X)_i = \sum_{x \in X} P(x, i) \quad (9)$$

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.

3.3 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.² 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:

$$y_H = \sum_{i=1}^a \sum_{j=1}^b G_{primary}(x)_i \cdot G_i(x)_j \cdot E_{i,j}(x) \quad (10)$$

¹While this is equivalent to using an L2 loss, we find that using the squared coefficient of variation requires less hyperparameter tuning, since it produces output on the same scale regardless of the number of experts.

² We have not found the need for deeper hierarchies.

Our metrics of expert utilization change to the following:

$$Importance_H(X)_{i,j} = \sum_{x \in X} G_{primary}(x)_i \cdot G_i(x)_j \quad (11)$$

$$Load_H(X)_{i,j} = \frac{Load_{primary}(X)_i \cdot Load_i(X^{(i)})_j}{|X^{(i)}|} \quad (12)$$

$Load_{primary}$ and $Load_i$ denote 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.

4 ADDRESSING MOE SYSTEM PERFORMANCE ISSUES

A major challenge that arises from a naive MoE implementation is the shrinking batch problem. 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. The shrinking batch problem causes computational inefficiency, as computational devices such as CPUs or GPUs tend to benefit from the increased ratio between the number of arithmetic operations and memory accesses which happens when large batch sizes are used. Our solutions to the shrinking batch problem involve finding ways to increase the original batch size, which we discuss in the following.

Combination Across Time Steps: In our language models, we apply the 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, as shown in Figure 2. Doing so increases the size of the input batch to the MoE layer by a factor of the number of time steps, which we denote by n_s .³

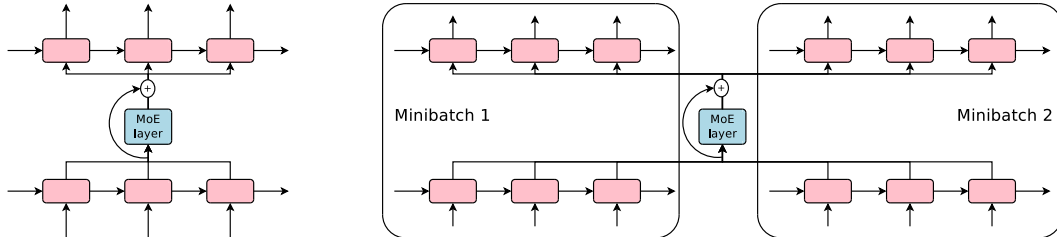


Figure 2: Left: Concatenate outputs of many time steps to form bigger expert batches. Right: Concatenate many minibatches from distributed data parallel models to form expert batches.

Combination Across Batches: In addition to combination across time steps, we employ a second technique which combines examples from different training batches, as shown in Figure 2-Right. In a conventional distributed training setting, multiple copies of the model on different devices process distinct batches of data, and parameter updates are performed asynchronously by a parameter server. In our technique, these different batches run synchronously. We distribute the standard layers of the model and the gating network according to the conventional data-parallel schemes but only keep 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. If the model is distributed over n_p devices, and each device receives a batch of size b , each expert receives a batch of approximately $\frac{k \cdot b \cdot n_s \cdot n_p}{n}$ examples. Thus, we achieve a factor of n_p improvement in expert batch size.

In the case of a hierarchical MoE (Section 3.3), the primary gating network employs data parallelism, secondary MoEs employ model parallelism, each one residing on its own device.

³We could not use this trick if we were applying the MoE recurrently between time steps, which might be a direction of future work.

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 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.

5 EXPERIMENTS

In the following, we present our experimental results on scaling up neural language model parameter sizes. We will show how these large models improve baselines in language modeling and neural machine translation.

For language modeling, we tested our method on the 1 Billion Word Language Model Benchmark (Chelba et al., 2013). We improved on the best published result for a single, word-level model (Jozefowicz et al., 2016), using a fraction of the computation.

Table 1: Summary of our results against state-of-the-art. For more details see Section 5.1

	Test Perplexity (10 epochs)	Test Perplexity (100 epochs)	#Parameters excluding embedding and softmax layers	Computation per Word excluding softmax layer	Training Time (10 epochs)
Our Fast Model	34.1		4303 million	8.9 million	15 hours on 16 k40s
Our Slow Model	28.0		4371 million	142.7 million	47 hours on 32 k40s
Best Published Results	34.7	30.6	151 million	151 million	59 hours on 32 k40s

For machine translation, we experimented with the WMT’14 En→Fr and En→De corpora, which have 36M sentence pairs and 5M sentence pairs, respectively. On these translation datasets, we compared our method against recently published Neural Machine Translation baselines in (Luong et al., 2015a; Zhou et al., 2016; Wu et al., 2016). We also experimented the performance of MoEs on Google’s internal translation data for English to French language pair, which is two orders of magnitudes larger than the WMT corpora (Wu et al., 2016). A summary of our BLEU score results, computation, and model parameter sizes against state-of-the-art is shown in Table 2. All our results are based on a single model. We achieved our results by significantly increasing the model parameter sizes while incurring far less computation per word.

Table 2: Summary of our results against state-of-the-art. For more details, see Tables 4, 5, and 7.

	WMT En→Fr	WMT En→De	Production En→Fr	Total #Parameters	Computation per Word
Ours	40.56	26.03	36.57	8690 million	100 million
Best Published Results	39.92	24.91	35.56	250 million	215 million

5.1 1 BILLION WORD LANGUAGE MODEL BENCHMARK

Dataset: We experimented with our method on the 1 Billion Word Language Model Benchmark, introduced by (Chelba et al., 2013). The dataset contains approximately 829 million words with a vocabulary of 793,471 words, including sentence boundary markers. The sentences are shuffled and duplicate sentences are removed. The words that are out of vocabulary are replaced by a special UNK token, which constitutes 0.3% of the dataset. The test data consists of 159,658 examples.

5.1.1 8-MILLION-OPERATIONS-PER-TIMESTEP MODELS

We investigate the effect of varying the number of experts in the MoE (and thereby the number of parameters) while maintaining a fixed, modest computational budget: about 8 million multiply-and-adds per training example per timestep in the forwards pass, excluding the softmax layer. More concisely, we call this "8 million ops/timestep". By comparison, the best published LSTM models described in (Jozefowicz et al., 2016) use 151 million ops/timestep.

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 dropout (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 3.1 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.

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 4. 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-parameter search to find the best dropout probability, in increments of 0.1.

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 3. For each model, we report the test perplexity, the computational budget, the parameter counts, the value of $DropProb$, and the training time.

All of the computationally-matched baseline models achieved similar test perplexities of 44.7 to 46.1. For the MoE models, test perplexity improved by 24% as the number of experts increased, from 45.0 for non-sparse 4-expert model to 34.1 for the 4096-expert model. These results are plotted in 3-left.

Conclusions: The result for the 4096-expert model seems hugely significant in that:

- The MoE layer demonstrates 99.9% sparsity (4 billion parameters, 4 million ops/timestep), without significant loss of GPU-based computational efficiency.
- It beats well-optimized computationally matched baselines by 24% perplexity.
- The best previously-published result for a word-level neural language model on this dataset is the 2xLSTM-8192-1024 model from (Jozefowicz et al., 2016), which achieves a perplexity of 34.7 after 10 training epochs and 30.6 after 100 training epochs (3 weeks). Our model beats the 10-epoch result for that model, using only 6% of the computation per example.

As can be seen in Figure 3-left, adding additional experts provides diminishing returns, especially after the number of parameters exceeds 1 billion. We hypothesize that this is due to the billion-word corpus size, and that larger corpora could benefit from even larger MoE layers. This hypothesis remains to be tested in future work.

Table 3: Model comparison on 1 Billion Word Language Model 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	Training Time 10 epochs
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	8h, 32 k40s
LSTM-2048-512	44.7		9.4	9.4	0.8	0.1	13h, 16 k40s
4xLSTM-512	46.0		8.4	8.4	0.8	0.1	13h, 16 k40s
MoE-1-Wide	46.1		8.4	8.4	0.8	0.1	13h, 16 k40s
MoE-1-Deep	45.7		8.4	8.4	0.8	0.1	12h, 16 k40s
MoE-4	45.0		8.4	8.4	0.8	0.1	18h, 16 k40s
MoE-32	39.7		8.4	37.8	0.9	0.1	12h, 16 k40s
MoE-256	35.7		8.7	272.9	1.1	0.1	14h, 16 k40s
MoE-256-h	36.0		8.4	272.9	1.1	0.1	15h, 16 k40s
MoE-1024-h	34.6		8.5	1079.0	1.9	0.2	14h, 16 k40s
MoE-4096-h	34.1		8.9	4303.4	5.1	0.2	15h, 16 k40s
2xLSTM-8192-1024*	34.7	30.6	151.0	151.0	1.8	0.25	59h, 32 k40s
MoE-34M	31.3		33.8	4313.9	6.0	0.3	17h, 32 k40s
MoE-143M	28.0		142.7	4371.1	6.0	0.4	47h, 32 k40s

5.1.2 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 3. 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%.

5.2 MACHINE TRANSLATION ON WMT’14 EN→FR AND EN→DE

Dataset: We benchmarked our method on the WMT’14 En→Fr and En→De corpora, where the 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

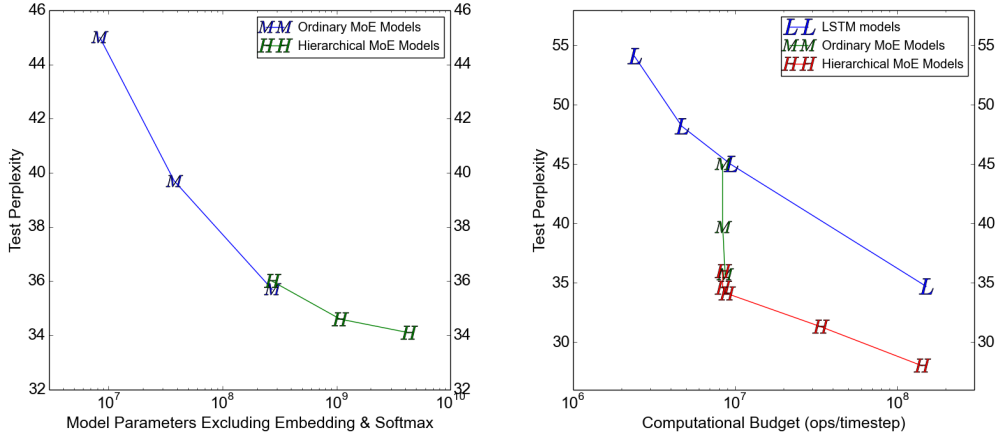


Figure 3: Model comparison on 1-Billion-Word Language-Modeling Benchmark. The graph on the left plots test perplexity against number of parameters for our 8-million-ops-per-timestep MoE models. These models form the nearly-vertical line at the center of the graph on the right, which plots test perplexity against computational budget. The top line represents the LSTM models from (Jozefowicz et al., 2016). The bottom line represents the three 4-billion parameter MoE models with varying computational budgets.

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.

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). We used an attention mechanism between the encoder and decoder, with the first decoder LSTM receiving output from and providing input for the attention.⁴ 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 added 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 used a model with a shared source and target vocabulary of 32K wordpieces. We also used the same beam search technique as proposed in (Wu et al., 2016).

Each MoE layer in our model is composed of up to 2048 experts. 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 used the strictly-balanced gating function described in Appendix A.

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. Similarly to (Wu et al., 2016) and to the previous section, 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 4.

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).

⁴For performance reasons, we used a slightly different attention function from the one described in (Wu et al., 2016) - See appendix B

Results: We first report the performance of our method as we increase the number of experts in our model. This set of results is shown in Figure 4-left. Each step approximately processes 200k words. 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. This supports the argument of using MoE for training large neural translation models.

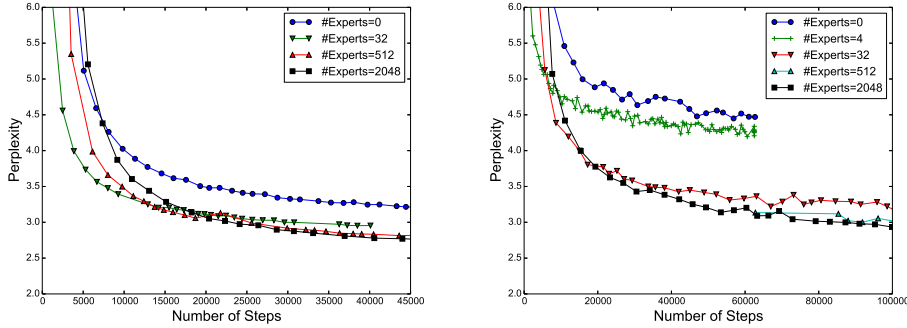


Figure 4: Perplexity on WMT’14n En→Fr dataset (left) and Google Production En→Fr dataset (right). All models are identical except for the number of experts in MoE. In all the models except for the one with no experts, the gating network selects exactly 4 experts to process each example.

The results of our method on En→Fr with 2048 experts and other baselines are shown in Table 4. As can be seen from the table, our approach achieved a BLEU score of 40.56. Since our method did not use RL refinement, this result provides a significant gain of 1.34 BLEU score on top of a strong baseline (Wu et al., 2016). The perplexity is also much better given the same size of vocabulary as described in (Wu et al., 2016). Note that our best model (i.e., MoE with 2048 experts) reached to the reported BLEU and perplexities much faster than the baseline while using fewer GPUs.

Table 4: Perplexity and BLEU comparison of our method against previous state-of-art method on WMT’14 En→Fr (newstest2014).

Model	Test Perplexity	Test BLEU	ops/timestep	Total #Parameters	Training Time
MoE with 2048 Experts	2.69	40.35	100.8M	8.690B	3 days/64 k40s
MoE with 2048 Experts (longer training)	2.63	40.56	100.8M	8.690B	6 days/64 k40s
GNMT (Wu et al., 2016)	2.79	39.22	214.2M	246.9M	6 days/96 k80s
GNMT+RL (Wu et al., 2016)	2.96	39.92	214.2M	246.9M	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 5: Perplexity and BLEU comparison of our method against previous state-of-art methods on WMT’14 En→De (newstest2014).

Model	Test Perplexity	Test BLEU	ops/timestep	Total #Parameters	Training Time
MoE with 2048 Experts	4.64	26.03	100.8M	8.690B	1 day/64 k40s
GNMT (Wu et al., 2016)	5.25	24.91	214.2M	246.9M	1 day/96 k80s
GNMT+RL (Wu et al., 2016)	8.08	24.66	214.2M	246.9M	1 day/96 k80s
PBMT (Durrani et al., 2014)		20.7			
DeepAtt (Zhou et al., 2016)		20.6			

The results of our method on En→De with 2048 experts and other baselines are shown in Table 5. As can be seen from the table, our method achieved a BLEU score of 26.03 which is 1.12 better than the best model in (Wu et al., 2016). The perplexity was also much better given the same size of vocabulary as described in (Wu et al., 2016).

We found that the experts indeed become highly specialized by syntax and/or semantics, as can be seen in Table 6. For example, one expert is used when the indefinite article “a” introduces the direct object in a verb phrase indicating importance or leadership.

Table 6: 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 ...
...	...	...

5.3 MACHINE TRANSLATION ON PRODUCTION EN→FR DATA

Dataset, Model Architecture, and Training: We also tested our method on our production English to French data, which is 2 orders of magnitude larger than the WMT’14 corpus. We used the exact same model architecture and training procedure as described in previous section.

Results: Figure 4-right demonstrates the significance of increasing the number of parameters in the model in order to improve learning. It shows the improvement in perplexity on Google Production En→Fr data on models with different number of experts, from a baseline of 0 expert to a MoE model of 2048 experts. If applicable, in all the models exactly 4 experts were used for processing each example. Each step approximately processes 200k words. As we increased the number of experts, from a baseline of 0 expert to the ones with 2048 experts the perplexity constantly improved. The improvement was achieved while the amount of computation per word was constant across all the models with non-zero experts.

Table 7: Perplexity and BLEU comparison of our method against previous state-of-art methods on the Google Production En→Fr dataset.

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	100.8M	8.690B	1 day/64 k40s
GNMT (Wu et al., 2016)	2.78	35.80	2.87	35.56	214.2M	246.9M	6 days/96 k80s

In Table 7, we report the comparison between our 2048-expert model and the strong and highly optimized baseline models in (Wu et al., 2016). Our model achieved 1.01 higher BLEU score on test data while speeding up computation by a factor of $6\times$ using fewer machines.

6 CONCLUSION

This work introduces Mixture-of-Experts (MoE) as a new type of layer in neural networks. The proposed approach trains an outrageously large network that contains thousands of experts and billions of parameters. We propose a sparse gating approach that enables training such a large network. We describe technical details on our data- and model- parallel system that is integral to MoE layer’s efficiency and scalability. On the 1 Billion Word Language Modeling Benchmark and WMT’14 translation benchmarks, our approach’s quality surpasses all currently published results. We also show that our approach can be applied to datasets with orders of magnitude more data, to deliver high quality language modeling and translation results. We find that increasing the model capacity well beyond the conventional approaches can lead to significant improvements on learning quality while incurring comparable or faster training time to the baseline models.

ACKNOWLEDGMENTS

We would like to thank all of the members of the Google Brain Team and the Google Translate Team who helped us with this project, in particular Geoffrey Hinton, Zhifeng Chen, Yonghui Wu, and Melvin Johnson.

REFERENCES

- 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.
- 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.
- Ronan Collobert, Samy Bengio, and Yoshua Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computing*, 2002.
- Marc Peter Deisenroth and Jun Wei Ng. Distributed Gaussian processes. In *ICML*, 2015.
- 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.
- Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 2000.
- 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.
- 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.
- 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.

- 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 Franoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH*, pp. 338–342, 2014.
- M. Schuster and K Nakajima. Japanese and Korean voice search. *ICASSP*, 2012.
- Babak Shahbaba and Radford Neal. Nonlinear models using dirichlet process mixtures. *JMLR*, 2009.
- Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.
- Lucas Theis and Matthias Bethge. Generative image modeling using spatial LSTMs. In *NIPS*, 2015.
- Volker Tresp. Mixtures of Gaussian Processes. In *NIPS*, 2001.
- Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, 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.

APPENDICES

A STRICTLY BALANCED GATING

Due to some peculiarities in our infrastructure which have since been fixed, at the time we ran 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. We expect that in a future version of this paper, we will re-run the machine translation experiments with Noisy-Top-K gating, and remove this appendix.

Recall that we define the softmax gating function to be:

$$G_\sigma(x) = \text{Softmax}(x \cdot W_g) \quad (13)$$

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

$$G(x)_i = \frac{G_\sigma(x)_i M(G_\sigma(x))_i}{\sum_{j=1}^n G_\sigma(x)_j M(G_\sigma(x))_j} \quad (14)$$

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

$$\text{TopK}(v, k)_i = \begin{cases} 1 & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ 0 & \text{otherwise.} \end{cases} \quad (15)$$

Batchwise Mask: To force each expert to receive the exact same number of examples, we introduce an alternative mask function, $M_{\text{batchwise}}(X, m)$, 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.

$$M_{\text{batchwise}}(X, m)_{j,i} = \begin{cases} 1 & \text{if } X_{j,i} \text{ is in the top } m \text{ values for expert } i \\ 0 & \text{otherwise} \end{cases} \quad (16)$$

As our experiments suggest and also observed in (Ioffe & Szegedy, 2015), using a batchwise function during training (such as $M_{\text{batchwise}}$) 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:

$$M_{\text{threshold}}(x, T)_i = \begin{cases} 1 & \text{if } x_i > T_i \\ 0 & \text{otherwise} \end{cases} \quad (17)$$

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.

$$L_{\text{batchwise}}(X, T, m) = \sum_{j=1}^{|X|} \sum_{i=1}^n (M_{\text{threshold}}(x, T)_i - M_{\text{batchwise}}(X, m)_{j,i})(X_{j,i} - T_i) \quad (18)$$

B 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:

$$A_{GNMT}(x_i, y_j) = \sum_{d=1}^n V_d \tanh((x_i U)_d + (y_j W)_d) \quad (19)$$

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:

$$A(x_i, y_j) = \sum_{d=1}^n V_d \tanh((x_i U)_d) \tanh((y_j W)_d) \quad (20)$$

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.