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

Presenters: Pratham Mehta & Rian Rahman

Problem Statement

- **Scaling Limits for Dense Neural Networks**
  - Deep Learning performance improves with increased model capacity
    - Capacity = Number of Parameters
  - Standard dense models activate all parameters for every input
  - As model size grows:
    - Computation cost increases very fast
    - Training time becomes prohibitive

$$\text{Compute per example} \propto \text{Model size}$$

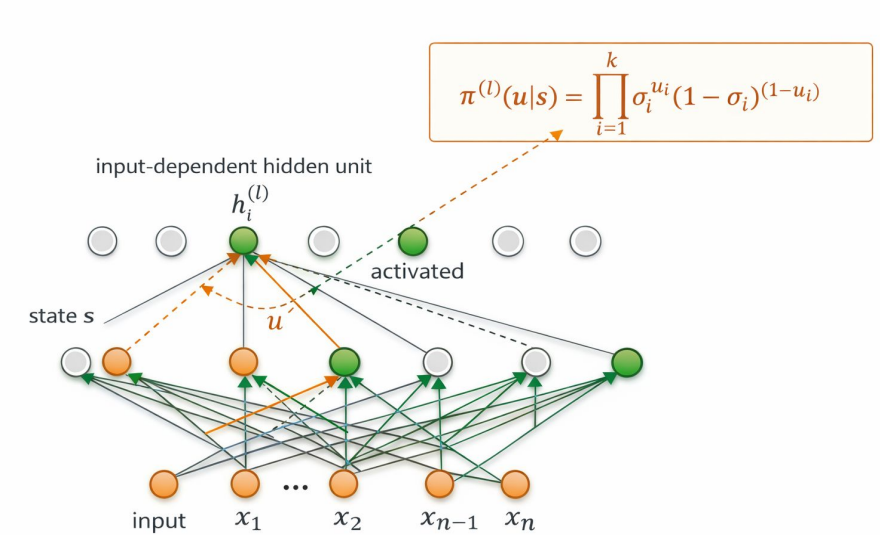
Problem Statement

- **Capacity vs Computational Bottleneck**
  - Increasing model size tightly couples:
    - Model capacity
    - Computation per example
  - For large scale training:
    - Total training cost scales with both model size and dataset size
  - Leads to near quadratic scaling of overall training cost
  - Hardware/distributed systems improvements cannot keep up
  - Result: scaling stalls despite having huge datasets

$$\text{Total training compute} \propto (\text{Model size}) \times (\text{Dataset size})$$

Motivation/Prior Work

- **Conditional Computation**
  - Conditional computation decomposes the model into two interacting parts:
    - A primary network responsible for computing predictions
      - Contains many potential computational paths
    - A gating or policy mechanism that decides which computation to perform
      - Selects subset of these paths based on the input



Motivation/Prior Work

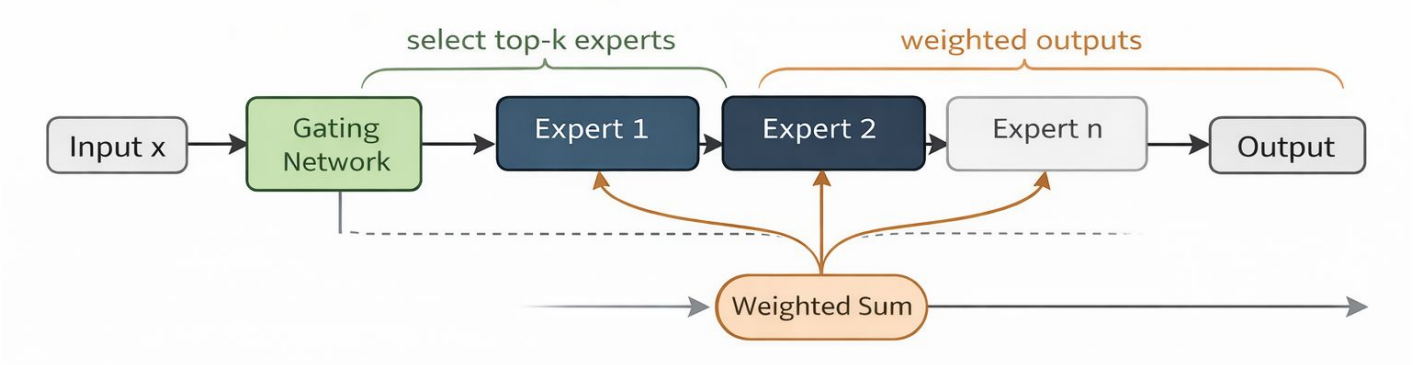
- **Conditional Computation (Continued)**
  - Conditional computation activates only parts of the network per input
  - Idea: increase capacity without proportionally increasing computation
  - Prior work explored:
    - Sparse Gating
    - Stochastic Routing
    - Reinforcement Learning Based Decisions
  - Promising in theory but not successful at large scale in practice

Motivation/Prior Work

- **Why Previous Conditional Computation Approaches Failed?**
  1. Mismatch with GPU Hardware
     - Modern GPUs are optimized for large dense matrix multiplications
     - Conditional execution introduces branching and irregular control flow
     - Leads to poor hardware utilization
  2. Shrinking Effective Batch Sizes
     - Conditional routing sends only a fraction of inputs to each expert
     - Each expert receives a much smaller batch
     - Small batches reduce efficiency
  3. Communication and Load Imbalance
     - Expert routing requires data movement across devices
     - Network bandwidth becomes a bottleneck in GPU clusters
     - Uneven expert usage causes load imbalance and unstable training

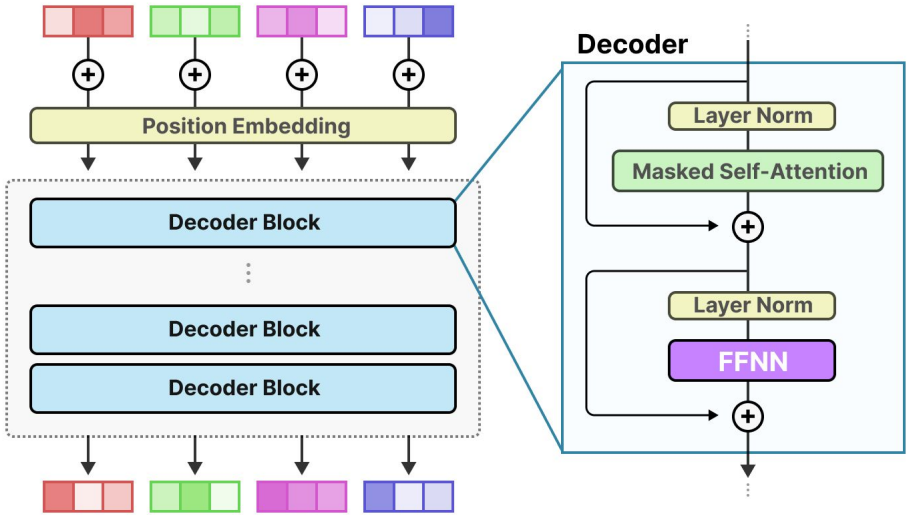
The Solution: Sparsely Gated Mixture of Experts Layer

- **Sparsely Gated Mixture of Experts (MoE) Layer**
  - Introduces a new general neural network component
  - Consists of:
    - A large set of expert networks
    - A trainable gating network
  - Gating network selects a sparse subset of experts per input
  - Entire system is trained end to end using backpropagation



The Solution: Sparsely Gated Mixture of Experts Layer

- **Sparsely Gated Mixture of Experts (MoE) Layer (Continued)**
  - The gating network routes each input to a small subset of experts
  - Each expert computes a candidate output for the input
  - Only selected experts are evaluated per example
  - Expert outputs are combined via a weighted sum



The Solution: Sparsely Gated Mixture of Experts Layer

- **Formal Mathematical Definition of the MoE output**
  - Let *G(x)* be the output of the gating network
  - Let *Ei(x)* be the output of the *i*-th expert
  - The MoE output is a weighted sum of expert outputs:

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

    - If *G(x)i = 0*, expert *Ei* is not computed
  - Sparsity directly reduces computation

The Solution: Sparsely Gated Mixture of Experts Layer

- **Softmax Gating Formulation**
  - A simple gating mechanism uses softmax over expert scores
  - Gating network computes:

$$G_\sigma(x) = Softmax(x \cdot W_g)$$

  - Produces a dense probability distribution over experts
  - Limitation:
    - All experts receive non-zero weight
    - Does not achieve computational sparsity

The Solution: Sparsely Gated Mixture of Experts Layer

- **Noisy Top K Gating for Sparse Routing**
  - Adds two key components to softmax gating:
    - Sparsity
    - Noise
  - First compute noisy logits:

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

  - Keep only the top *k* values:

$$G(x) = Softmax(KeepTopK(H(x), k))$$

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

  - Sparsity saves computation
  - Noise helps prevent expert collapse and improves load balancing

Key Technical Challenges and Solutions

- **The shrinking batch problem**:
  - Modern CPUs/GPUs need large batches for computational efficiency
  - If the gating network chooses k out of n experts for each example, then each expert receives a much smaller batch
  - For a batch of examples b, each expert sees

$$\frac{kb}{n} \ll b$$

  - This makes a naive MoE computational inefficient on GPUs
  - Solution: Make the original batch size as large as possible
  - However the batch size is limited by memory
  - The authors propose techniques to increase the batch size

Key Technical Challenges and Solutions

- **Mixing Data Parallelism and Model Parallelism**:
  - Solution: change the distributed training scheme.
    - Standard layers + gating → data parallel
    - Experts → model parallel (single shared copy)
    - Experts receive merged batches from all devices
  - If model runs on d devices, each expert now sees:

$$\frac{kbd}{n}$$

  - This helps achieve an improvement by a factor of d in expert batch size
  - This also restores large effective batch sizes and allows scaling experts linearly with hardware.

Key Technical Challenges and Solutions

- **Network Bandwidth Constraint**:
  - In distributed GPU clusters:
    - Compute capacity ≫ Network bandwidth
  - Sending inputs/outputs of experts over network is the bottleneck
  - Solution:
    - Use very large hidden layers inside experts.
  - Because:
    - Computation ≫ Communication
  - keeps GPUs compute-bound, not network-bound.
  - Authors note that due to this, we can increase computational efficiency simply by using a larger hidden layer, or more hidden layers.

Key Technical Challenges and Solutions

- **Expert Collapse Problem**:
  - Without constraints, gating converges to a few experts.
  - This is self-reinforcing:
    - Selected experts train faster
    - Become better
    - Get selected more
  - They define Importance of an expert over a batch:

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

  - Then add loss:

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

  - This forces uniform utilization of all experts

Key Technical Challenges and Solutions

- **Load Imbalance Problem**:
  - Even with equal importance, experts may receive different numbers of examples.
  - They introduce Load loss:

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

  - Which prevents memory and performance problems on distributed hardware.
  - These two losses are critical for stable MoE training.

Evaluation

- **1B Word Language Modeling Benchmark:**
  - **Hypothesis**: Massive capacity via MoE improves quality at constant compute.
  - **Goal**: To test if MoE models could absorb vast amounts of knowledge from very large text corpora and achieve superior perplexity scores.
  - **Dataset**: shuffled unique sentences from news articles, totaling approximately 829 million words, with a vocabulary of 793,471 words..
  - **Model Architecture**:
    - 2 LSTMs with MoE in between
    - Up to 4096 experts
    - Only 4 active per input
    - Same compute: ~8M ops/timestep
    - Low computation, varied capacity

Evaluation

- **Result: Capacity Improves Perplexity at Same Compute**

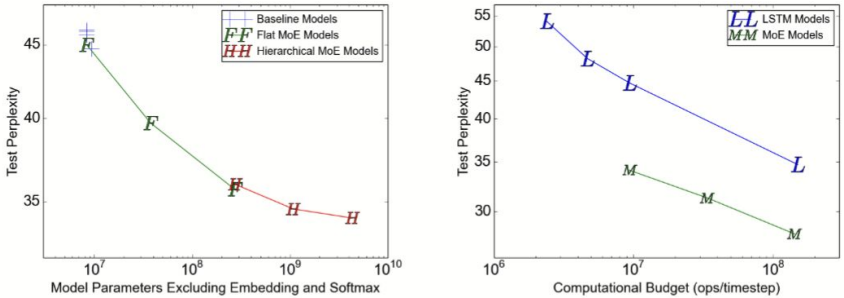


Figure 2: Model comparison on 1-Billion-Word Language-Modeling Benchmark. On the left, we plot test perplexity as a function of model capacity for models with similar computational budgets of approximately 8-million-ops-per-timestep. On the right, we plot test perplexity as a function of computational 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.

  - The largest model (4096 experts) achieved 24% lower perplexity.
  - Models had identical compute, but parameter count varied massively.
  - This proves that quality scales with capacity, not computation, when sparsity is used.

Evaluation

- **High Capacity vs SOTA LSTMs**

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**

  - They trained 4 billion parameter models
  - Better perplexity than best published LSTMs
  - Requires only 6% of the computation
  - Even the fastest MoE model beats the best published result.

Evaluation

- **100B Word Corpus Scaling**

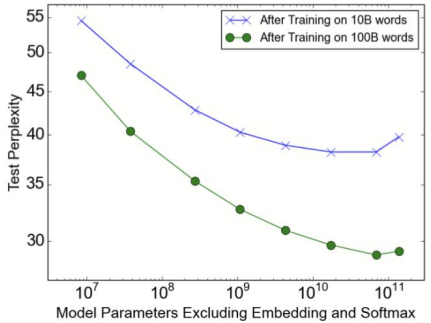


Figure 3: Language modeling on a 100 billion word corpus. Models have similar computational budgets (8 million ops/timestep).

  - Scale to:
    - 131,072 experts
    - 137B parameters
    - 99.994% sparsity
  - Result: 39% lower perplexity than compute-matched LSTM baseline.
  - Even at 65536 experts (99.994% layer sparsity), computational efficiency for the model stays at a respectable 0.72 TFLOPS/GPU.
  - Benefit increases with dataset size.

Evaluation

Machine Translation Results

- MoE added to GNMT:
  - +1.34 BLEU (En→Fr)
  - +1.12 BLEU (En→De)
  - Multilingual MoE beats multilingual GNMT on 11/12 pairs
  - Trains faster
- Shows MoE works beyond language modeling.

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

Evaluation

Machine Translation Results

- MoE added to GNMT:
  - +1.34 BLEU (En→Fr)
  - +1.12 BLEU (En→De)
  - Multilingual MoE beats multilingual GNMT on 11/12 pairs
  - Trains faster
- Shows MoE works beyond language modeling.

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

Contextualization

- **Why MoE Matters Beyond This Paper?**
  - MoE as a general scaling primitive
    - Decouples model capacity from per example computation
    - Enables scaling parameter count without linear compute growth
  - Broader impact
    - Foundation for large scale language models
    - Enables specialization across experts

Limitations

- **Remaining Challenges**
  - Expert specialization is not guaranteed
    - Experts may converge to similar functions without additional regularization
  - Routing introduces optimization instability
    - Discrete top-k selection creates noisy gradients
    - Training can become sensitive to initialization and hyperparameters
  - Communication overhead remains a bottleneck
    - Expert routing requires data movement across devices
    - Network bandwidth limits scalability at large expert counts

Conclusion

- **Key Takeaways**
  - This work is the first practical success of conditional computation.
  - They showed:
    - How to scale models to 1000× capacity
    - How to keep computation constant
    - How to make MoE train stably on GPUs
    - How to overcome batch, bandwidth, and load issues
  - This paper is the ancestor of modern MoE LLMs
  - Core ideas still used today:
    - Top-K gating
    - Expert parallelism
    - Load balancing losses