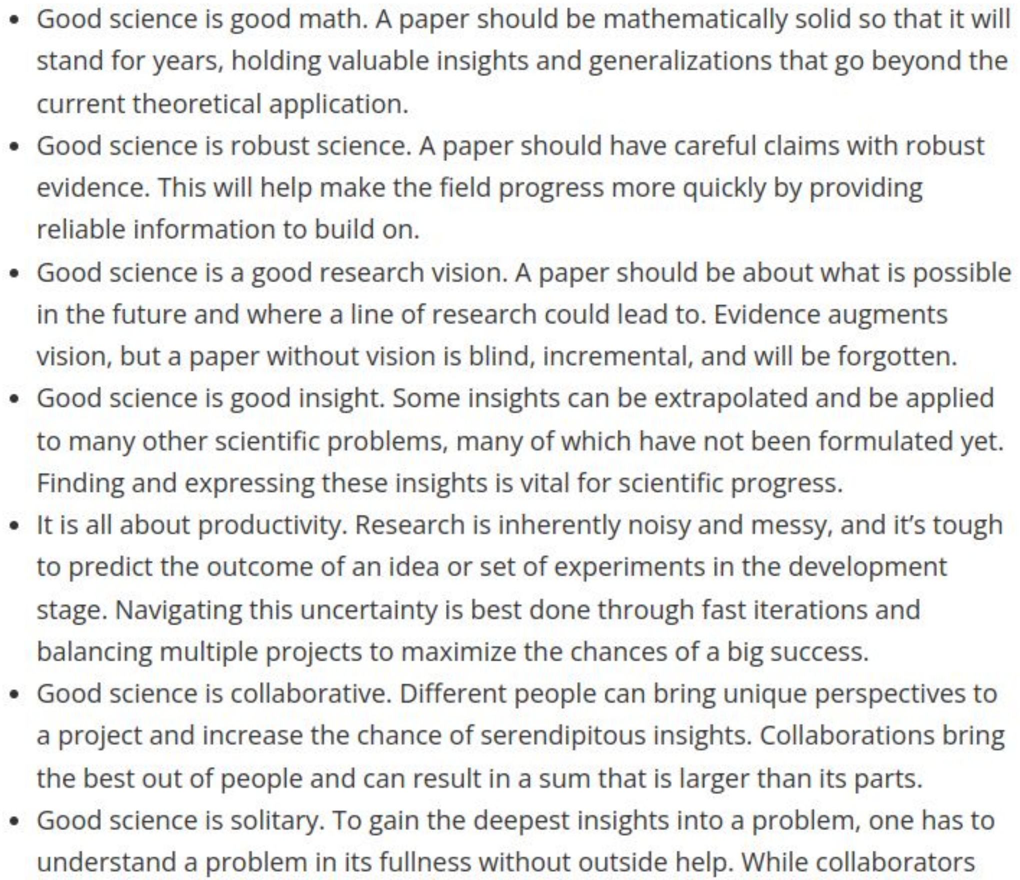
Beyond transformers: Mixture of Expert Models

CSE517: Grad NLP
Tim Dettmers
2023-02-16

What makes a good researcher?

- Top 1% of researcher produce 20% of all citations
- Accumulated advantage the most important factor that creates good researchers
- How Nobel Laureates are made (according to Harriet Zuckerman):
  - Nobel Laureates start out not better than any other researcher
  - They gain a small advantage through chance or luck
  - That advantage gives them more resources and opportunities (“better” university, more grants, more students, awards, more lab resources, more GPUs)
  - Once they receive the advantage, it's easier to gain more advantage
  - Repeat … repeat … repeat …
  - Nobel Laureate
- By this point, Nobel Laureates are better researcher, mostly due to research style learned from their advisor and research group

What makes a good scientist? Research style!



- Good science is good math. A paper should be mathematically solid so that it will stand for years, holding valuable insights and generalizations that go beyond the current theoretical application.
- Good science is robust science. A paper should have careful claims with robust evidence. This will help make the field progress more quickly by providing reliable information to build on.
- Good science is a good research vision. A paper should be about what is possible in the future and where a line of research could lead to. Evidence augments vision, but a paper without vision is blind, incremental, and will be forgotten.
- Good science is good insight. Some insights can be extrapolated and be applied to many other scientific problems, many of which have not been formulated yet. Finding and expressing these insights is vital for scientific progress.
- It is all about productivity. Research is inherently noisy and messy, and it's tough to predict the outcome of an idea or set of experiments in the development stage. Navigating this uncertainty is best done through fast iterations and balancing multiple projects to maximize the chances of a big success.
- Good science is collaborative. Different people can bring unique perspectives to a project and increase the chance of serendipitous insights. Collaborations bring the best out of people and can result in a sum that is larger than its parts.
- Good science is solitary. To gain the deepest insights into a problem, one has to understand a problem in its fullness without outside help. While collaborators

This lecture is about following a particular research style

You will learn in this lecture about mixture of experts, but that is not the main lesson.

The main lesson is to follow the research style of “computational efficiency is everything” to its conclusion.

Our tenet for this lecture

Intelligence = min(Compute/waste, data movement/waste)

Important:

- Balance computation and data movement (communication)
- Maximize computation and data movement to create the most intelligence
- Reduce waste to a minimum: Every computation and data movement needs to be needed

Not important:

- Anything that has not to do with these variables

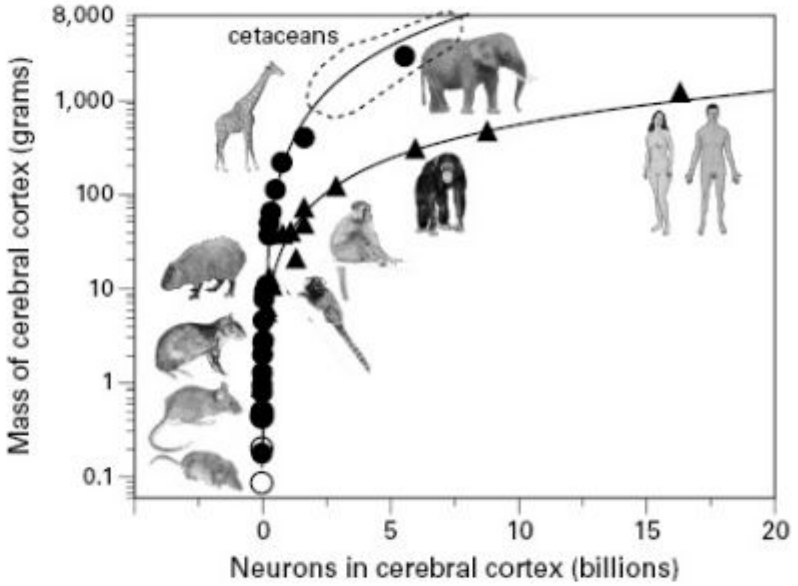
This is naive in many ways. It is a hypothetical view

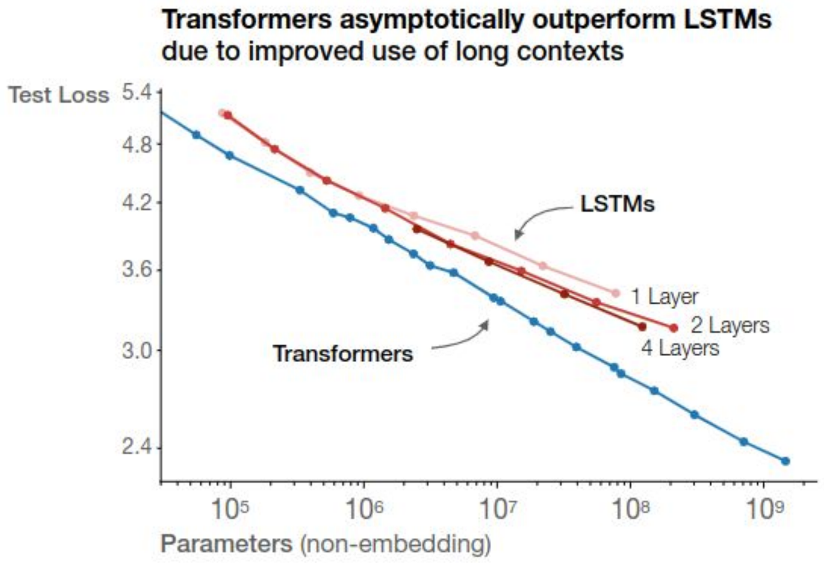
- Each research style or view has different believes
- Believes often do not rely on good evidence or have some weak points
- Opposite research styles might be successful
- Some research styles are only successful for a while to focus on a particular problem

Why the computational perspective could make sense

1. Intelligence roughly proportional to number of cortical neurons (compute):
    a. Humans 10x more neurons than Chimpanzee/Dolphins/Crows/Elephants
    b. Chimpanzees 10x more neurons than dogs
    c. Dogs 2x more neurons than cats
    d. Cats 50x more neurons than bees
    e. GPT-3 roughly as much compute as a bee
2. Primate brains are special. They are optimized for data movement (sparsity)
3. Human brains not special. Just a scaled up monkey brain.
4. Communication (white matter) and computation (grey matter) mathematically perfectly balanced across all primates (no waste)
5. Number of neurons determined by maximum energy intake across all mammals (no waste)

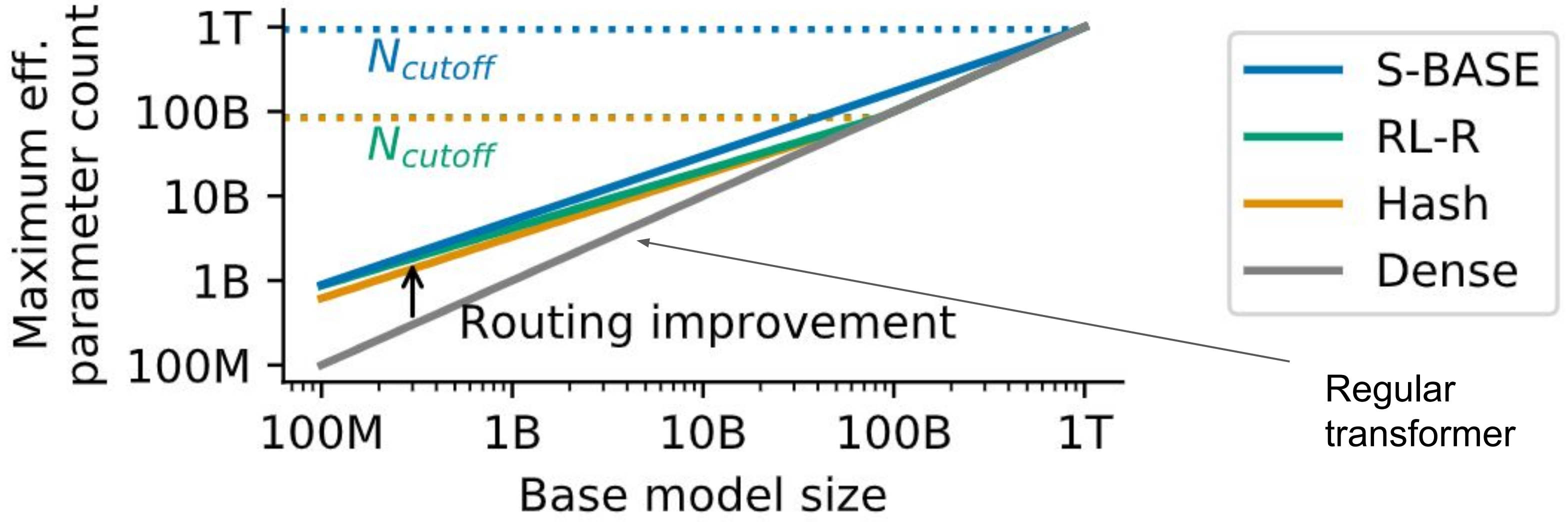
Primates scale differently … and so do transformers





(left) Herculano-Houzel, 2016. (right) Kaplan et al. 2020.

What scales better than transformers? Mixture of Experts



Regular transformer

Unified Scaling Laws for Routed Language Models. Clark et al., 2022.

Background

In this section you will learn what limits us: compute/data movement, are we too wasteful?

What is the main problem, computation or data movement?

Why are GPUs fast?

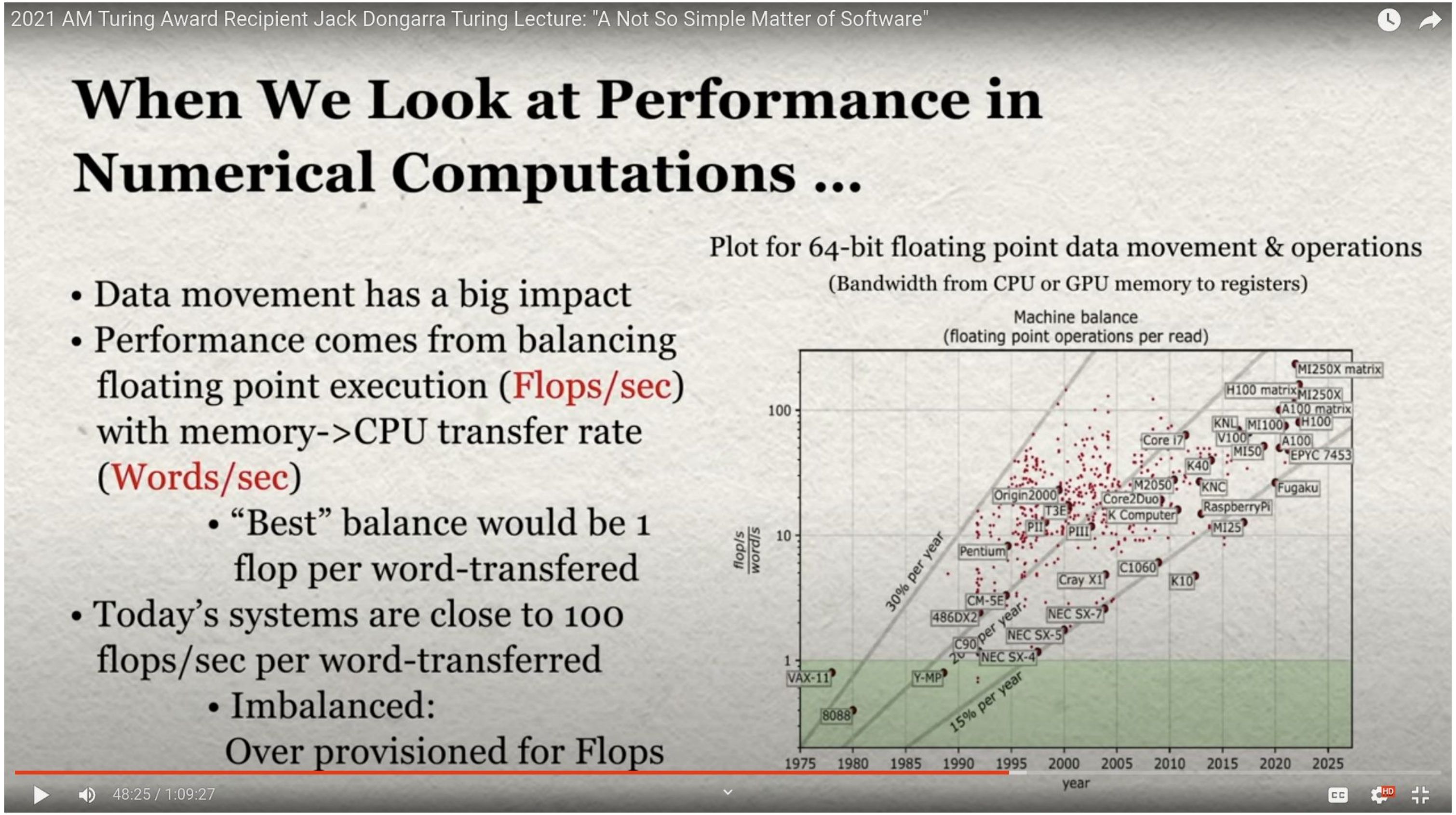
How to do a fast matrix multiplication that balances compute and data movement?

How do we scale efficiently?

What are language models and how do they work?

What are transformers and how do they work?

Communication (data movement) much more limiting than computation



2021 AM Turing Award Recipient Jack Dongarra Turing Lecture: "A Not So Simple Matter of Software"

When We Look at Performance in Numerical Computations ...

- Data movement has a big impact
- Performance comes from balancing floating point execution (Flops/sec) with memory->CPU transfer rate (Words/sec)
    - "Best" balance would be 1 flop per word-transfered
- Today's systems are close to 100 flops/sec per word-transferred
    - Imbalanced: Over provisioned for Flops

Plot for 64-bit floating point data movement & operations

(Bandwidth from CPU or GPU memory to registers)

How GPUs overcome data limitations, and why they are fast

Task: pick up data (packages) from location A (main memory).

Packages go from A (GPU memory) to B (GPU core). The core has a loading dock (cache) that is for packages that just arrived. Its small and cannot hold many packages at once.

CPUs are like Ferraris, super fast, but can only hold a few packages

GPUs are like trucks, super slow, but can hold lots of packages

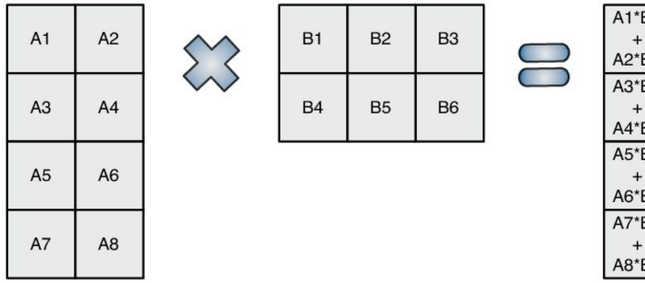
What is more efficient for package delivery, a fleet of Ferraris or a fleet of trucks?

Caching in matrix multiplication I

Most efficient to divide a matrix multiplication into smaller sub-matrix multiplications.

How to do matmul efficiently with three trucks:

1. Truck A, B, and C drive to core.
2. Truck A and B unload packages (A1, A2) and (B1, B4) unto the loading dock and get more B packages (B2, B5, B3, B6).
3. Truck C waits in the loading dock.
4. Workers from the core grab packages A and B and matrix multiply them. Then store results into Truck C. Once Truck C is full, truck C drives to the output
5. The other trucks arrive with packages B2, B5, B3, B6
6. Repeat …



Caching in matrix multiplication II

In matrix multiplication A*B=C we:

Multiply each row of A with ***all columns*** from B.

As such, memory of A can be reused as often as there are columns in B.

Multiply each column of B with ***all rows*** of A

As such, memory of B can be reused as often as there are rows in B

Chinchilla scaling laws and how to estimate compute time

Chinchilla scaling laws (Hoffmann et al., 2022) say we should use 20 training tokens per parameter.

So if our dataset has 380B tokens (The Pile) we need a ~20B model to be efficient. Training transformers incurs 6 FLOPs per parameter per token. So we have:

380e9 * 20e9 *6 FLOPs = 4.56e22 FLOPs

A A40 GPU on Hyak has 150 TFLOPS or 150e12 FLOPS performance. At 25% efficiency we have 38e12 FLOPS.

4.56e22 / 3.8e13 / (60*60*24*365) = 38.05 GPU years

So to optimally train a model on 380B tokens, we need 38 GPU years.

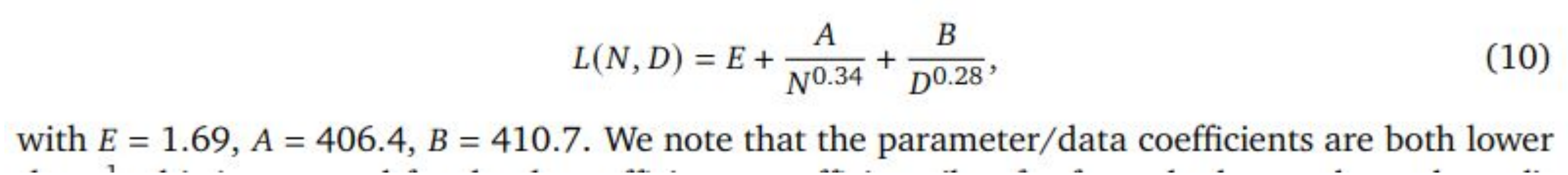
Calculating the waste of GPT-3 training: I

GPT-3 data set had 300B tokens and model size was 175B:

300e9*175e9*6 = 3.15e23 FLOPs

Chinchilla optimal for 300B tokens is 15B parameters

300e9*15e9*6 = 2.27e22 FLOPs



$$L(N, D) = E + \frac{A}{N^{0.34}} + \frac{B}{D^{0.28}}, \qquad (10)$$

with $E = 1.69$, $A = 406.4$, $B = 410.7$. We note that the parameter/data coefficients are both lower

Calculating the waste of GPT-3 training: II

We use Chinchilla scaling law and increase data and compute equally until we reach the loss for GPT-3.

Solution: 32B parameters on 638B tokens to reach GPT-3 performance.

638e9*32e9*6 = 1.25e23 FLOPs

GPT-3 waste: 3.15e23 / 1.25e23 = 2.52

GPT-3 could have been trained to the same performance with 2.52x less compute.

Putting things together

According to the Chinchilla paper (Hoffmann et al., 2022) one needs 20 tokens per parameter to train in the most cost efficient way.

The internet has about 5T tokens and about 2T of “high quality” tokens.

Max effective model size is 250 - 100B parameters.

For 250B - 100B parameters, Mixture of Experts are about 3-5x more efficient.

Chinchilla scaling is 2.5x more efficient compared to GPT-3 training.

-> Chinchilla mixture of expert training is 8x to 13x more efficient than GPT-3 training

Language modeling

Objective: Take a sequence of words and predict the next word.

Why is this a good objective:

- Learn grammar and style first
- Then model the world. You need to model people in order to understand that they could say/write next -> emotional state, political affiliation, social status, social situation, current time (past/future/present), fictional or non-fictional

Transformers

Example: “Her dog felt guilty. Daisy looked down avoiding eye contact. She was fully aware of the ___”

How to think about transformers. A simplification:

1. Create word embeddings. Each word has many “features”. Each feature is an association of what is going on for this word: grammar, time, emotional state, social situation, environment, animate vs inanimate.
2. Combine ***association for a word*** to create new condensed association for that word
3. Weighted sum ***across tokens for each associations*** to create new context-enriched associations

Mixture of Experts

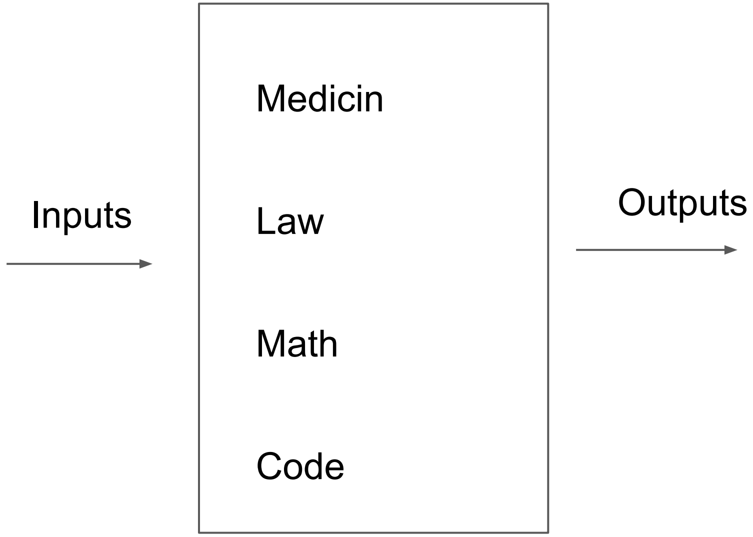
Transformers vs mixture of experts

Some associations are mutually independent. Associations of heart medicine not very useful to answer questions about astrophysics. Computing heart medicine associations on astrophysics data is a waste of compute.

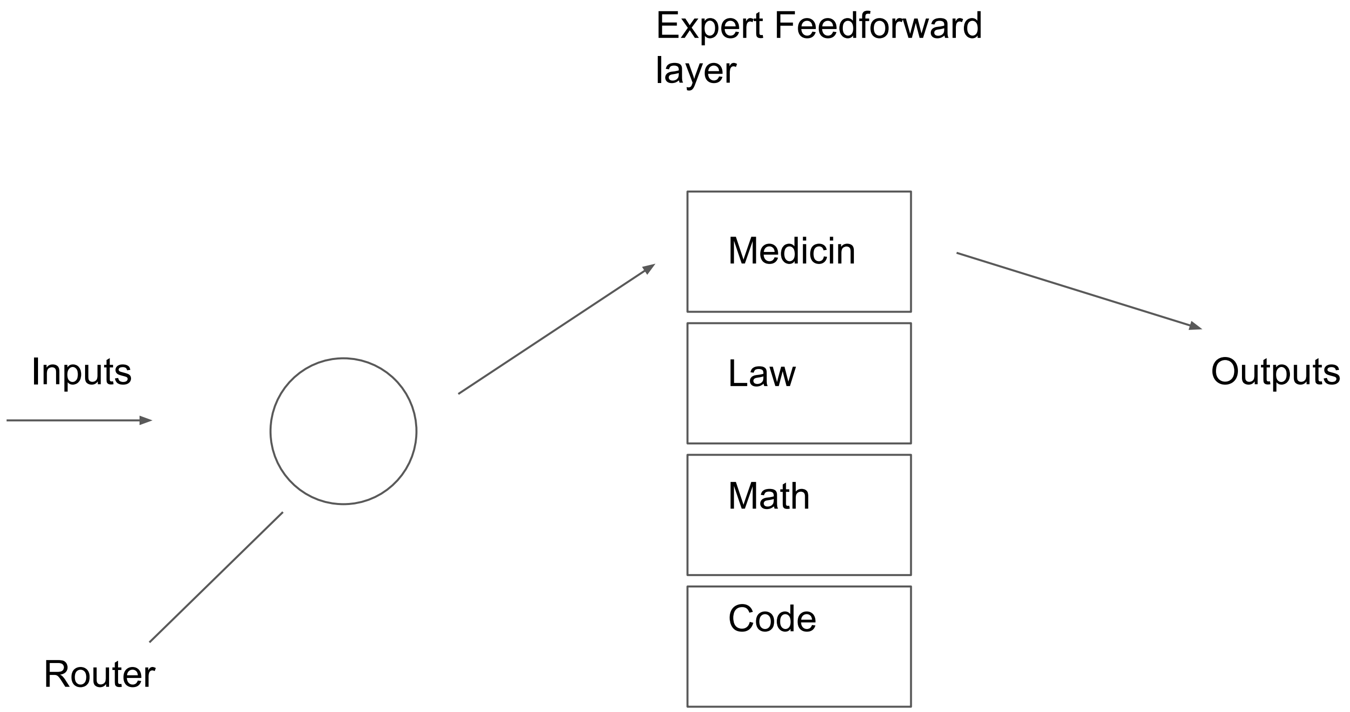
Can we arrange associations into distinct packages that activate when needed rather than being activated during any context?

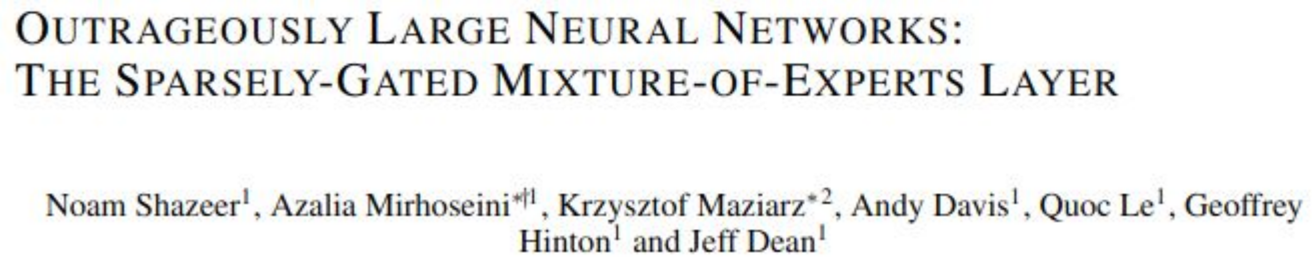
The Transformer is wasting compute

Transformer
Feedforward layer



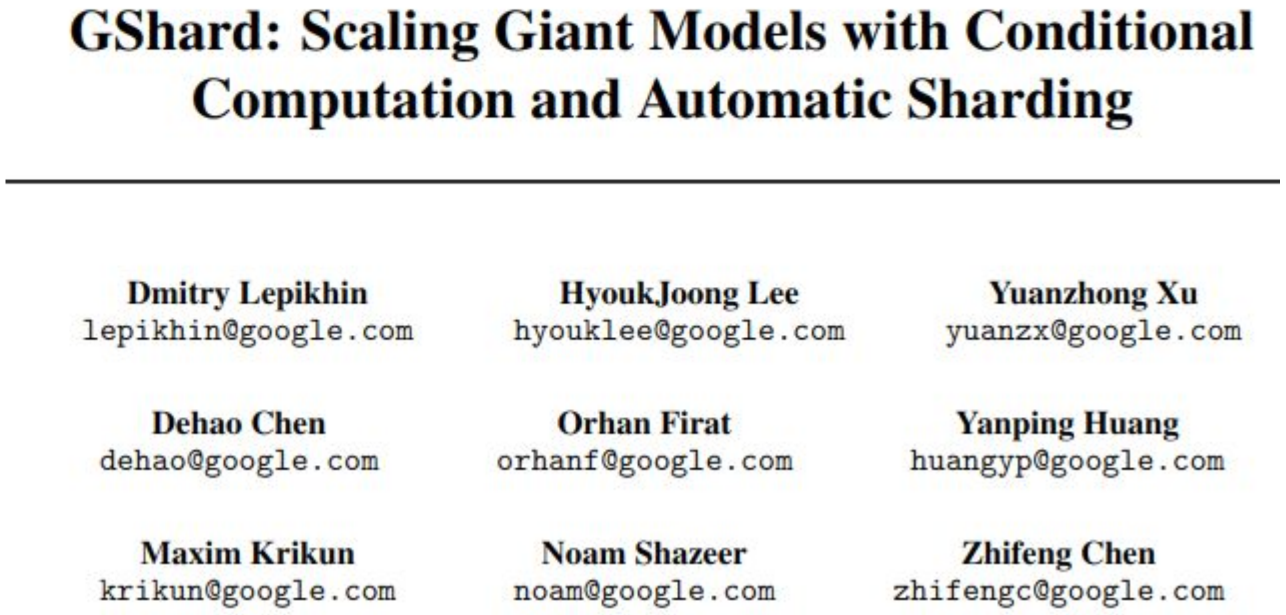
Expert perspective





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]



GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding

Dmitry Lepikhin
lepikhin@google.com

HyoukJoong Lee
hyouklee@google.com

Yuanzhong Xu
yuanzx@google.com

Dehao Chen
dehao@google.com

Orhan Firat
orhanf@google.com

Yanping Huang
huangyp@google.com

Maxim Krikun
krikun@google.com

Noam Shazeer
noam@google.com

Zhifeng Chen
zhifengc@google.com

GShard

Three parts:

1. Expert architecture
2. Extra loss function to avoid degeneration
3. Load-balancing to avoid waiting for slowest expert

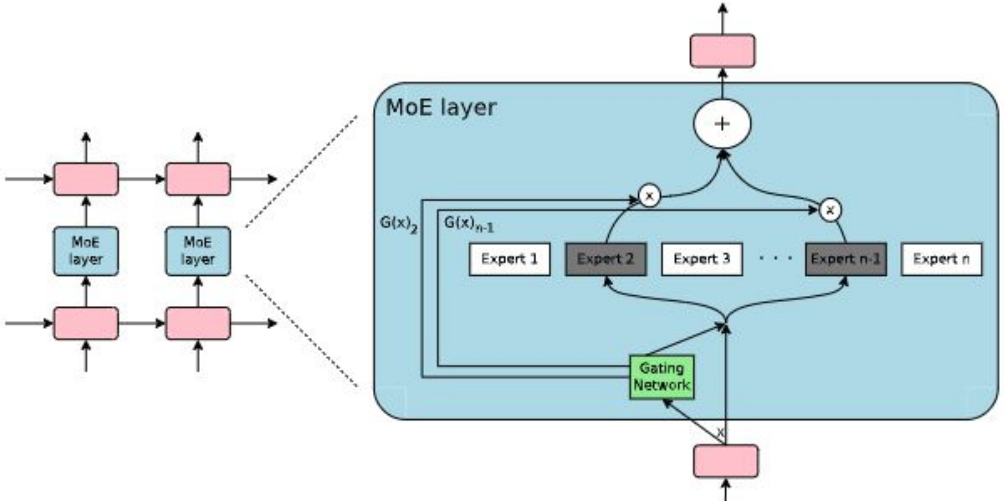
GShard: Architecture

For each token

1. Select expert via top-k softmax $G(x) = Softmax(KeepTopK(H(x), k))$
2. Pass through expert, then multiply by softmax value

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

$$\begin{aligned} \mathcal{G}_{s,E} &= \mathrm{GATE}(x_s) \\ \mathrm{FFN}_e(x_s) &= wo_e \cdot \mathrm{ReLU}(wi_e \cdot x_s) \\ y_s &= \sum_{e=1}^{E} \mathcal{G}_{s,e} \cdot \mathrm{FFN}_e(x_s) \end{aligned}$$



GShard: Auxiliary loss function

Problem: at the start of training the expert that is best might be selected again and again which leads to degeneration. As such, we penalize the distribution of experts.

L = loss + auxiliary loss

L_aux = num_experts * fraction of tokens per expert * mean(gate probability)

Expert has mean probability close to 0 -> 0 loss

Expert has mean probability close to 1/2E -> E*1/2E*1/2E = 1/4E loss

Expert has mean probability close to 1/E -> E*1/E*1/E = 1/E loss

Expert has mean probability close to 2/E -> E*2/E*2/E = 4/E loss

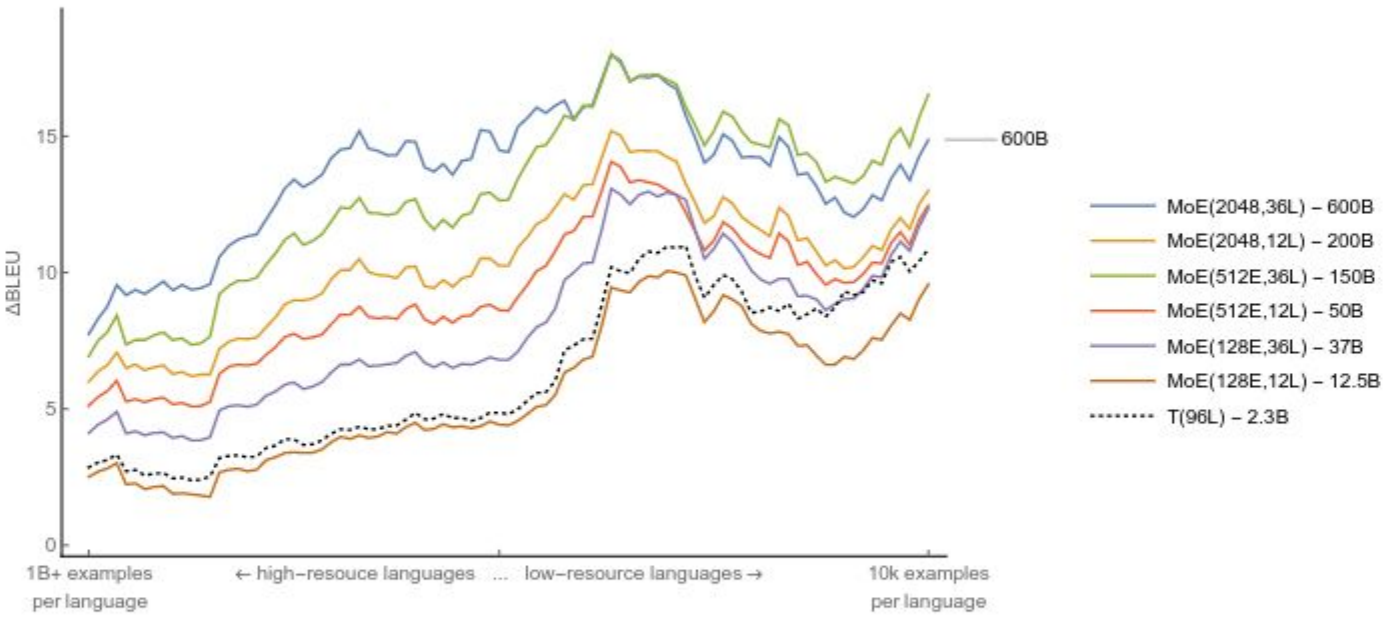
Expert has mean probability close to 1 -> E*E*1 = E^2 loss

Gshard: Load balancing

Problem: If we compute multiple experts in parallel with a different amount of tokens, some experts may have more tokens than others and take longer -> ***All experts need to wait for the slowest expert***.

Solution: We have a capacity limit of ~total_tokens/num_experts. Any tokens going beyond this limit will be set to zero and will not be routed to any expert.

Results

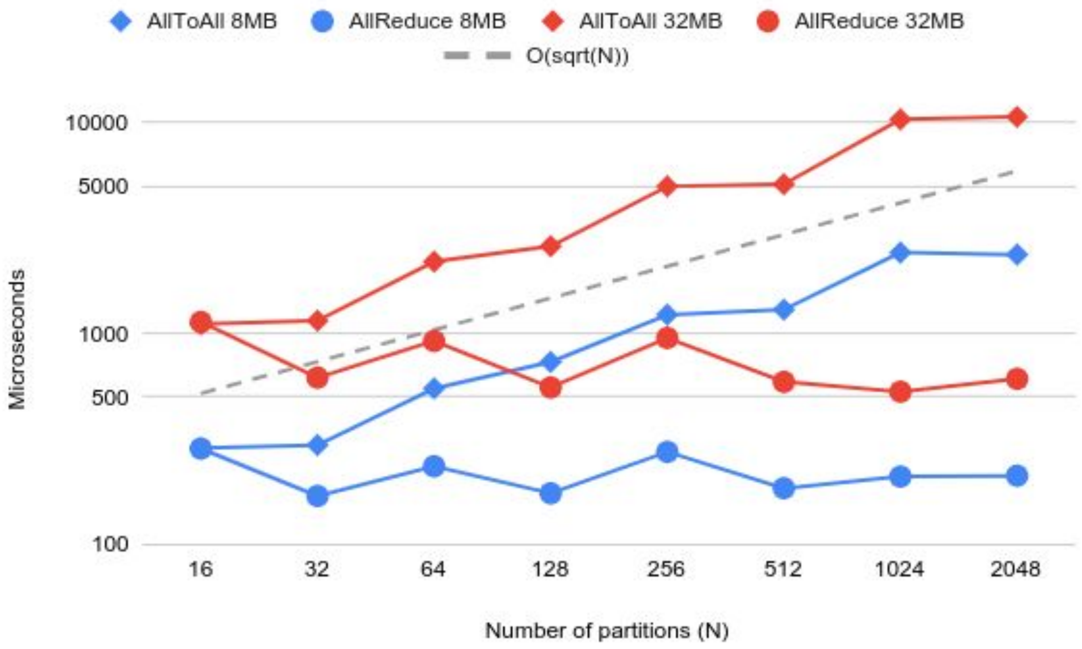


ΔBLEU
0
5
10
15
600B
1B+ examples per language
← high-resouce languages ... low-resource languages →
10k examples per language
MoE(2048,36L) – 600B
MoE(2048,12L) – 200B
MoE(512E,36L) – 150B
MoE(512E,12L) – 50B
MoE(128E,36L) – 37B
MoE(128E,12L) – 12.5B
T(96L) – 2.3B

Issues: Complicated and slow communication

- Extra loss
- Load-balancing via overflow

Can we do better?



BASE Layers: Simplifying Training of Large, Sparse Models

Mike Lewis[1] Shruti Bhosale[1] Tim Dettmers[1 2] Naman Goyal[1] Luke Zettlemoyer[1 2]

UNIFIED SCALING LAWS FOR ROUTED LANGUAGE MODELS

Aidan Clark*, Diego de las Casas*, Aurelia Guy*, Arthur Mensch*

Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman‡, Trevor Cai, Sebastian Borgeaud, George van den Driessche, Eliza Rutherford, Tom Hennigan, Matthew Johnson‡, Katie Millican, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Jack Rae, Erich Elsen, Koray Kavukcuoglu, Karen Simonyan

DeepMind Google Research‡

BASE layers approach

1. Randomly assign tokens, then rerank locally
2. Learn a centroid for each expert, compute similarity score between tokens and centroids
3. Use similarity score to rerank tokens to experts (linear assignment problem)

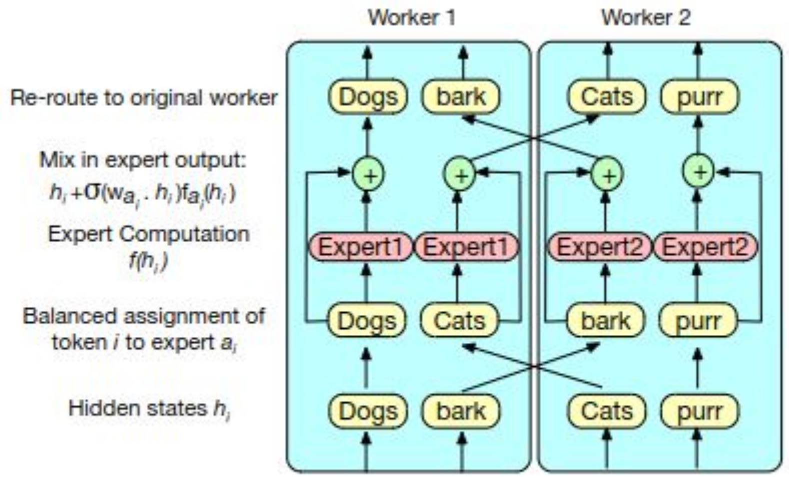

Worker 1
Worker 2
Re-route to original worker
Mix in expert output:
$h_i + \sigma(w_{a_i} \cdot h_i) f_{a_i}(h_i)$
Expert Computation
$f(h_i)$
Balanced assignment of token i to expert a_i
Hidden states h_i
Dogs
bark
Cats
purr
Expert1
Expert2

Figure 1. Overview of a BASE layer. Each worker contains a separate *expert* module. During training, we compute a balanced assignment of tokens such that each worker sends an equal number of tokens to each expert. By softly mixing in the expert module, experts can learn to specialize for particular types of tokens.

BASE layers: architecture

Regular transformer layers + BASE layers instead of feedforward network for every other layer.

h = input token, f_a = expert, w_a = expert_centroid, sigma = logistic sigmoid

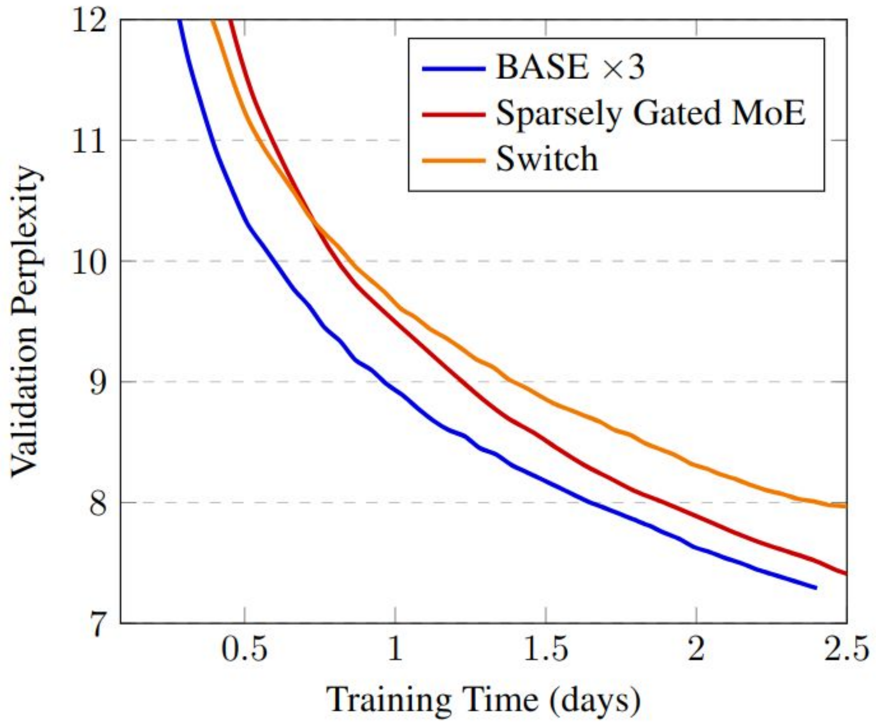
t = index into sequence (token)

$$\sigma(h_t \cdot w_{a_t}) f_{a_t}(h_t) + h_t,$$

Training: Assign tokens equally to each experts

Evaluation: Assign tokens to highest scoring expert

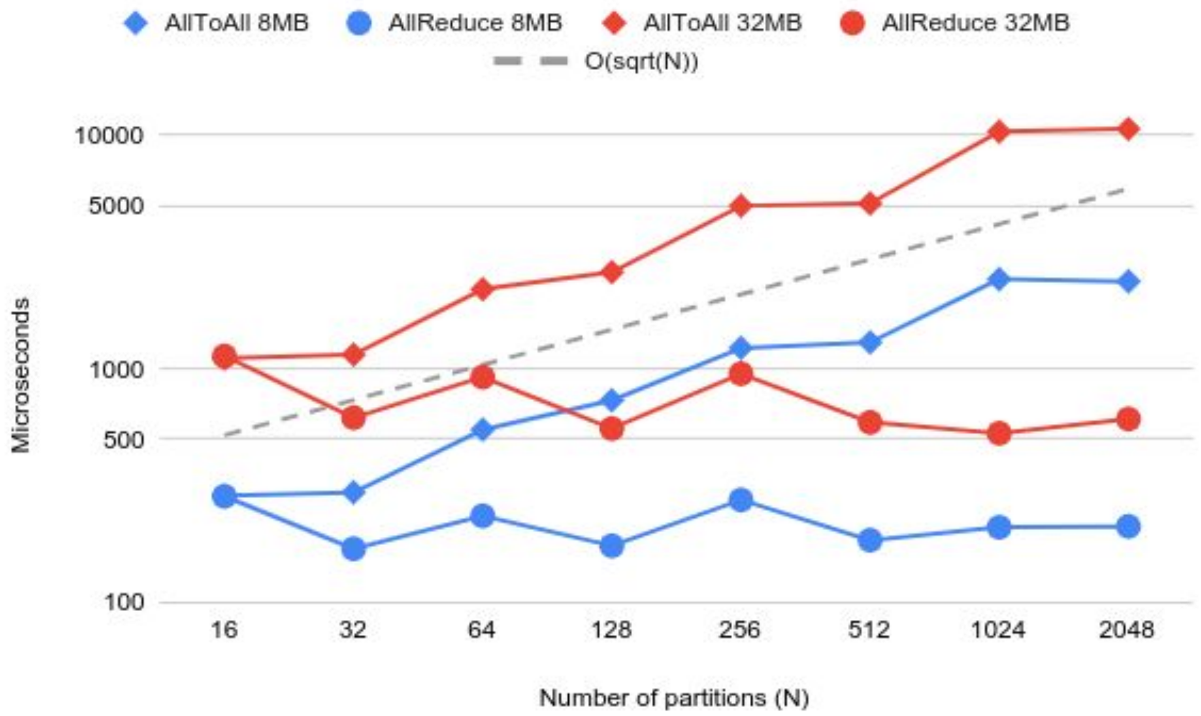
BASE Layers: Results



Model	Tokens per Second
Data Parallel	600k
Model Parallel ×2	224k
Sparsely Gated MoE	292k
Switch	469k
BASE	545k
BASE ×2	475k

Table 2. Number of tokens processed per second during training by different models. BASE computes updates faster than other approaches that divide models over multiple workers, due to reduced communication overheads. This allows a 43B parameter model to be trained at 90% of the speed of a 1.5B data parallel baseline.

Remaining issue: Slow communication

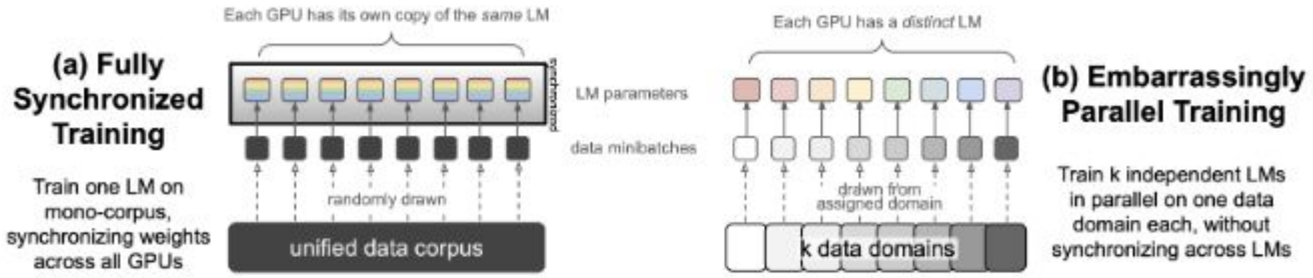


Branch-Train-Merge: Embarrassingly Parallel Training of Expert Language Models

Margaret Li[*†◇] **Suchin Gururangan**[*†◇] **Tim Dettmers**[†]

Mike Lewis[◇] **Tim Althoff**[†] **Noah A. Smith**[†♠] **Luke Zettlemoyer**[†◇]

[†]Paul G. Allen School of Computer Science & Engineering, University of Washington
[♠]Allen Institute for AI
[◇]Meta AI


Each GPU has its own copy of the *same* LM
(a) Fully Synchronized Training
synchronized
LM parameters
data minibatches
randomly drawn
unified data corpus
Train one LM on mono-corpus, synchronizing weights across all GPUs
Each GPU has a *distinct* LM
drawn from assigned domain
k data domains
(b) Embarrassingly Parallel Training
Train k independent LMs in parallel on one data domain each, without synchronizing across LMs

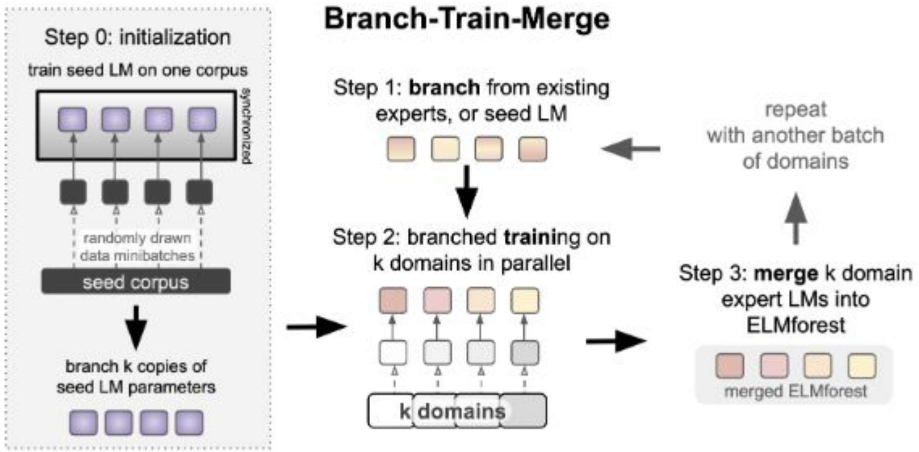
Branch-Train-Merge: Architecture

It’s just a transformer! So no changes to the architecture or no new loss.

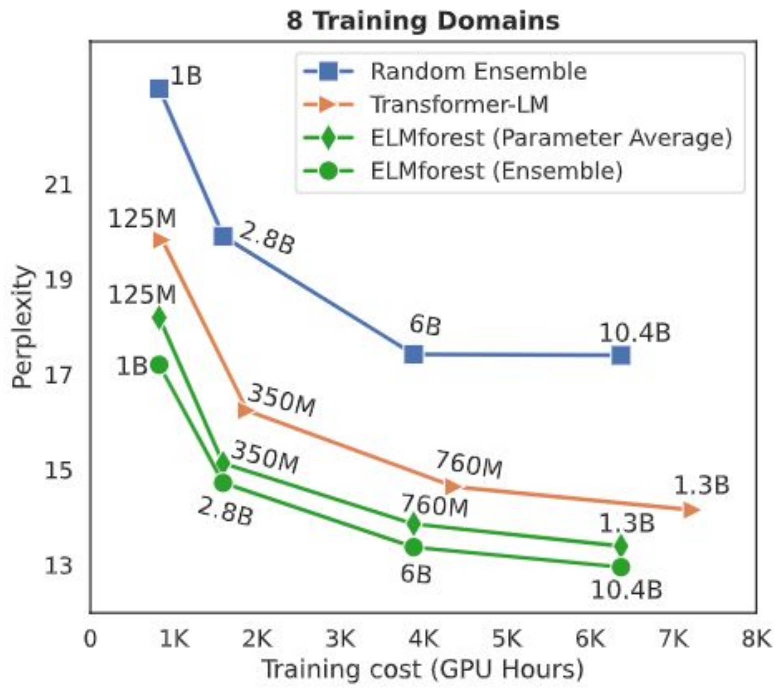
Branch-Train-Merge: Training

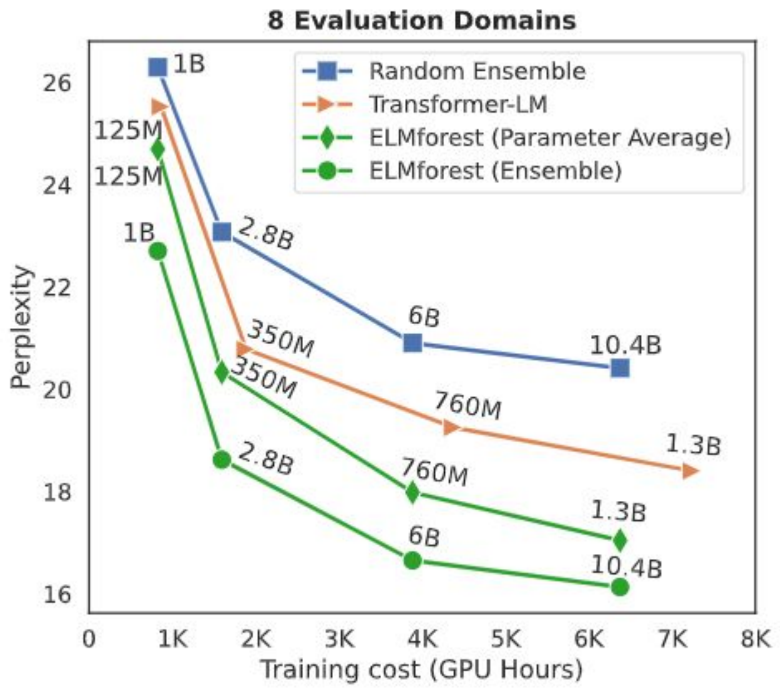
How to train:

- Branch: Take a reference LLM or train new one for a couple thousand steps
- Branch: Copy its weights for each domain
- Train: Feed data of each domain through its designated "expert"
- Train: Train all experts independently (no communication needed)
- Merge: Merge experts into a single model. This is done through evaluation on a validation set.

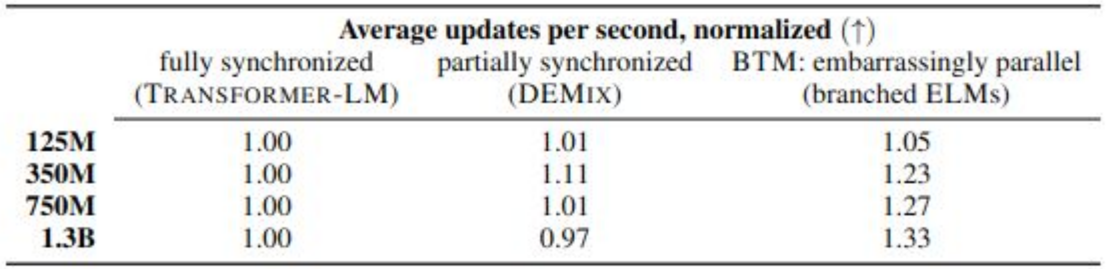

Step 0: initialization
train seed LM on one corpus
synchronized
randomly drawn data minibatches
seed corpus
branch k copies of seed LM parameters
Branch-Train-Merge
Step 1: branch from existing experts, or seed LM
Step 2: branched training on k domains in parallel
k domains
repeat with another batch of domains
Step 3: merge k domain expert LMs into ELMforest
merged ELMforest

Branch-Train-Merge: Results. Better than transformers





Branch-Train-Merge: Results. Faster than transformers.



	Average updates per second, normalized (↑)		
	fully synchronized (TRANSFORMER-LM)	partially synchronized (DEMIX)	BTM: embarrassingly parallel (branched ELMs)
125M	1.00	1.01	1.05
350M	1.00	1.11	1.23
750M	1.00	1.01	1.27
1.3B	1.00	0.97	1.33

Conclusion

If we want to maximize min(compute/waste, data movement/waste) then:

1. GPUs balance compute and data movement
2. Scaling laws help us reduce waste in our training setup
3. Mixture of experts help us to reduce waste by reducing unneeded computation
    a. BASE layers balances compute and data movement compared to GShard
    b. Branch-train-merge eliminates data movement across experts entirely

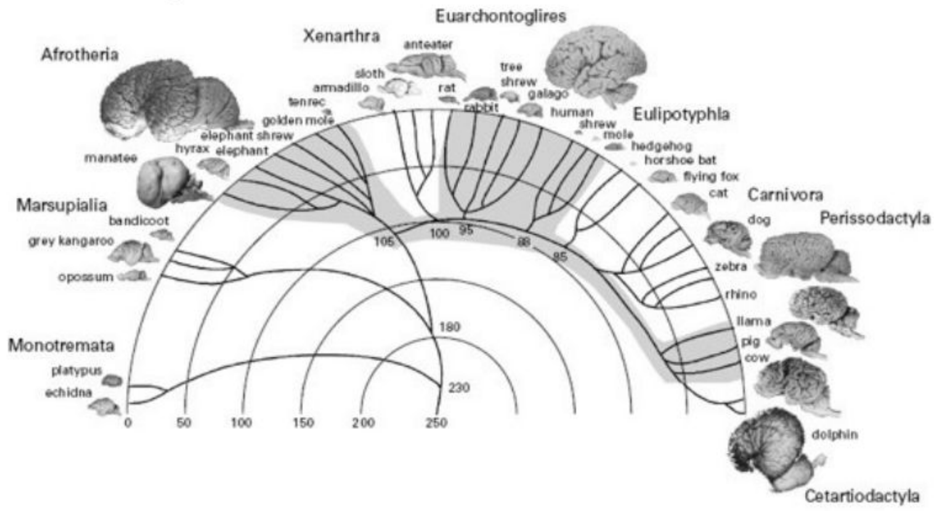
Thank you!
Questions / Comments?

Why does this tenet make sense: Evidence from neuroscience

Study brains of all mammals.

Finding (1): brain structure correlated with evolutionary tree structure

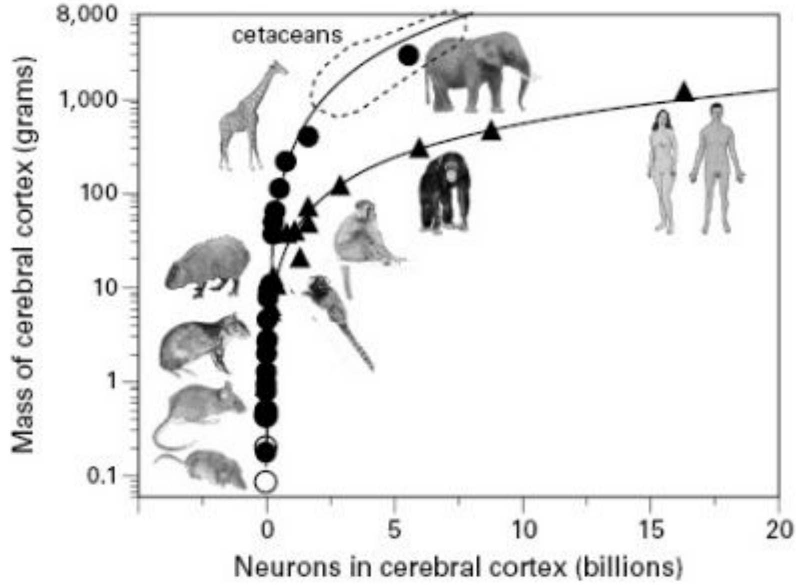
he last 300 million years of evolution.



Why does this tenet make sense: Humans have the most compute

Finding (2): Intelligence ~ number of neurons

Finding (3): Primate neurons scale differently

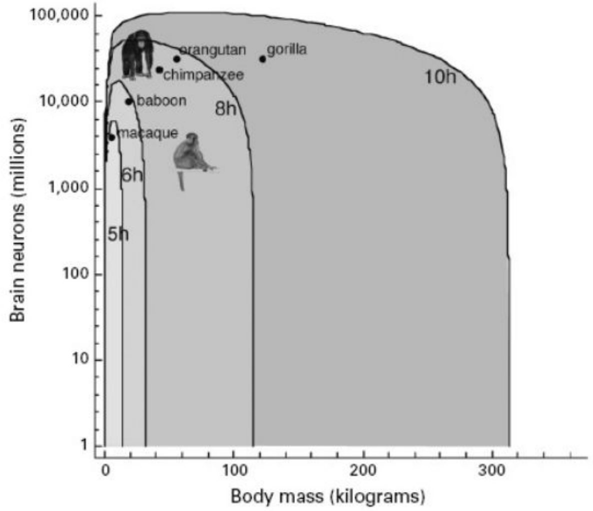


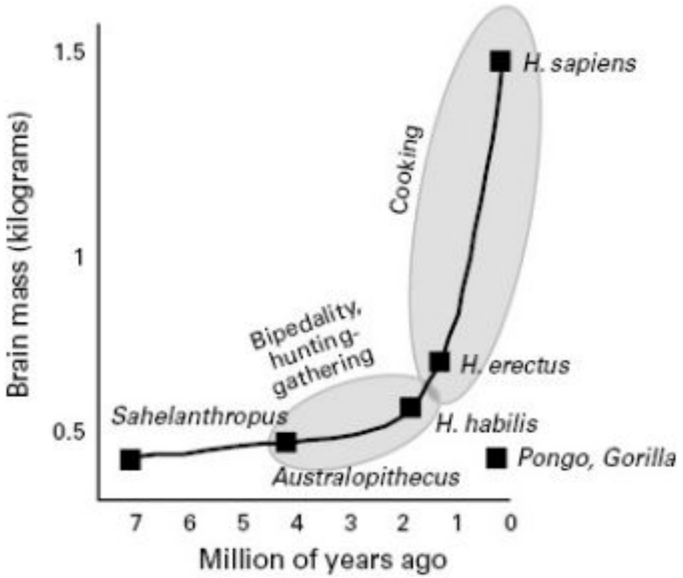
- A cat has 50x more neurons than a bee
- A dog has 2x more neurons than a cat
- A dolphin/elephant/chimpanzee/crow has 5x more neurons than a dog
- A humans has 10x more neurons than a chimpanzees

Why does this tenet make sense: Humans produce the most energy

Finding (4): Intelligence is energy limited: Chose 1: (a) large body, (b) large brain

Finding (5): Humans are more energy efficient and can afford more intelligence.


Brain neurons (millions)
Body mass (kilograms)
orangutan
gorilla
chimpanzee
baboon
macaque
10h
8h
6h
5h


Brain mass (kilograms)
Million of years ago
H. sapiens
Cooking
Bipedality, hunting-gathering
H. erectus
H. habilis
Sahelanthropus
Australopithecus
Pongo, Gorilla