

Mixture of Experts

UCSB LLM Seminar

Claire Murphy

UC Santa Barbara

September 4, 2026



Introduction

Review: The Transformer Architecture

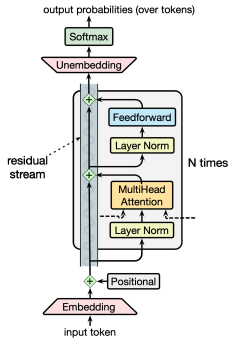


Figure: Figure from [JM].

Autoregressive LLM: Given a prompt, how do we predict the next token in the sequence?

- 1 Convert the prompt into an $N \times d$ dimensional matrix, where each row corresponds to a token.
- 2 Pass the prompt through several layers of **transformer blocks**; we obtain an $N \times d$ dimensional matrix.
 - 1 Layer norm: Normalizing vectors
 - 2 Multihead attention: The heart of the Transformer computation
 - 3 Feed forward layer: Neural network
- 3 Use the **language modeling head** to produce a probability distribution over all possible tokens. Use this distribution to predict the next token.

Mixture of Experts

MoE layers have a certain number of “experts” (e.g. 8), where each expert is a neural network.

Mixture of Experts

MoE layers have a certain number of “experts” (e.g. 8), where each expert is a neural network.

In practice, the experts are FFNs, but they can also be more complex networks or even a MoE itself, leading to hierarchical MoEs!

Mixture of Experts

MoE layers have a certain number of “experts” (e.g. 8), where each expert is a neural network.

In practice, the experts are FFNs, but they can also be more complex networks or even a MoE itself, leading to hierarchical MoEs!

Each separate network, or expert, specializes in a different region of the input space. A gating network determines the weights for each expert. During training, both the expert and the gating are trained.

Mixture of Experts

MoE layers have a certain number of “experts” (e.g. 8), where each expert is a neural network.

In practice, the experts are FFNs, but they can also be more complex networks or even a MoE itself, leading to hierarchical MoEs!

Each separate network, or expert, specializes in a different region of the input space. A gating network determines the weights for each expert. During training, both the expert and the gating are trained.

Switch Transformer: Replace feedforward layer with a sparse Mixture of Experts (MoE) layer.

Mixture of Experts cont.

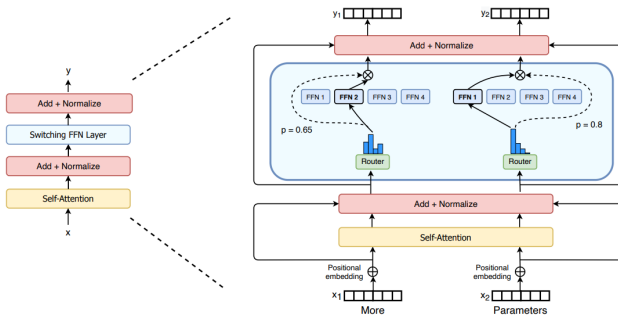


Figure: Within a Transformer block, we replace a single feedforward layer with a MoE layer. Figure from [STS].

History of MoE

1991 Jacobs et al., “Adaptive Mixtures of Local Experts” [JJN].

- 1 Each expert specializes in a different region of the input space.
- 2 A gating network determines the weights for each expert.
- 3 During training, both the expert and the gating are trained.

History of MoE

1991 Jacobs et al., “Adaptive Mixtures of Local Experts” [JJN].

- 1 Each expert specializes in a different region of the input space.
- 2 A gating network determines the weights for each expert.
- 3 During training, both the expert and the gating are trained.

2017 Shazeer et al., “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer” [SMM].

- 1 **Sparsity**: Only a few experts are chosen at a time, significantly decreasing the computational complexity.
- 2 Achieved “greater than 1000x improvements in model capacity with only minor losses in computational efficiency on modern GPU clusters.”

History of MoE

1991 Jacobs et al., “Adaptive Mixtures of Local Experts” [JJN].

- 1 Each expert specializes in a different region of the input space.
- 2 A gating network determines the weights for each expert.
- 3 During training, both the expert and the gating are trained.

2017 Shazeer et al., “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer” [SMM].

- 1 **Sparsity**: Only a few experts are chosen at a time, significantly decreasing the computational complexity.
- 2 Achieved “greater than 1000x improvements in model capacity with only minor losses in computational efficiency on modern GPU clusters.”

2020 Lepikhin et al., “GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding” [LLX].

- 1 Introduced **expert capacity** to ensure that all experts received a similar number of tokens.
- 2 This idea “enabled us to scale up multilingual neural machine translation Transformer model with Sparsely-Gated Mixture-of-Experts beyond 600 billion parameters.”

History of MoE

1991 Jacobs et al., “Adaptive Mixtures of Local Experts” [JJN].

- 1 Each expert specializes in a different region of the input space.
- 2 A gating network determines the weights for each expert.
- 3 During training, both the expert and the gating are trained.

2017 Shazeer et al., “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer” [SMM].

- 1 **Sparsity**: Only a few experts are chosen at a time, significantly decreasing the computational complexity.
- 2 Achieved “greater than 1000x improvements in model capacity with only minor losses in computational efficiency on modern GPU clusters.”

2020 Lepikhin et al., “GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding” [LLX].

- 1 Introduced **expert capacity** to ensure that all experts received a similar number of tokens.
- 2 This idea “enabled us to scale up multilingual neural machine translation Transformer model with Sparsely-Gated Mixture-of-Experts beyond 600 billion parameters.”

2021 Fedus et al., “Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity” [FZS].

Sparse Mixture of Experts

Motivation (from [SMM])

“When datasets are sufficiently large, increasing the capacity (number of parameters) of neural networks can give much better prediction accuracy.”

Motivation (from [SMM])

“When datasets are sufficiently large, increasing the capacity (number of parameters) of neural networks can give much better prediction accuracy.”

“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.”

Motivation (from [SMM])

“When datasets are sufficiently large, increasing the capacity (number of parameters) of neural networks can give much better prediction accuracy.”

“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.”

Conditional computing: “A way to increase model capacity without a proportional increase in computational costs.”

Some Notation

Some notation for MoE with n experts:

- 1 $x \in \mathbb{R}^d$: Token embedding received by MoE layer
- 2 $y \in \mathbb{R}^d$: Token embedding outputted by MoE layer
- 3 $G(x) \in \mathbb{R}^n$: Weights for each of the n experts
- 4 $E_i(x) \in \mathbb{R}^d$: Output of i th expert

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

Some Notation

Some notation for MoE with n experts:

- 1 $x \in \mathbb{R}^d$: Token embedding received by MoE layer
- 2 $y \in \mathbb{R}^d$: Token embedding outputted by MoE layer
- 3 $G(x) \in \mathbb{R}^n$: Weights for each of the n experts
- 4 $E_i(x) \in \mathbb{R}^d$: Output of i th expert

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

Sparsity: For only $k \ll n$ experts, $G_i(x) \neq 0 \Rightarrow$ we only need to compute $E_i(x)$ for k -many experts.

An Example

Some notation for MoE with n experts:

- 1 $x \in \mathbb{R}^d$: Token embedding received by MoE layer
- 2 $y \in \mathbb{R}^d$: Token embedding outputted by MoE layer
- 3 $G(x) \in \mathbb{R}^n$: Weights for each of the n experts
- 4 $E_i(x) \in \mathbb{R}^d$: Output of i th expert

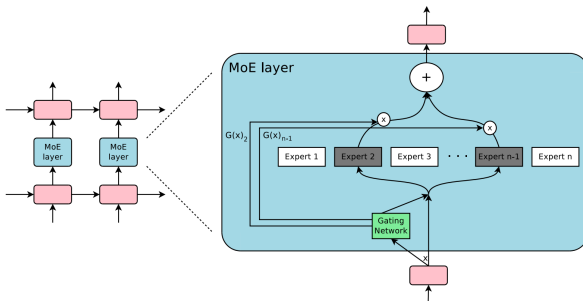


Figure: In this example, we select $k = 2$ experts with weights $G(x)_2$ and $G(x)_{n-1}$. Figure from [SMM].

The Gating Mechanism

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

Naive / non-sparse: For trainable weight matrix W_g , compute $G(x) = \text{softmax}(xW_g)$.

The Gating Mechanism

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

Naive / non-sparse: For trainable weight matrix W_g , compute $G(x) = \text{softmax}(xW_g)$.

Noisy Top-k Gating: Introduces some (tunable) noise and then keeps the top k values:

1 $H(x)_i := (xW_g)_i + Z\sigma((xW_{\text{noise}})_i)$, where

1 $Z \sim \mathcal{N}(0, 1)$.

2 σ is the “softplus” activation function, $\sigma(s) = \ln(1 + e^s)$.

The Gating Mechanism

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

Naive / non-sparse: For trainable weight matrix W_g , compute $G(x) = \text{softmax}(xW_g)$.

Noisy Top-k Gating: Introduces some (tunable) noise and then keeps the top k values:

1 $H(x)_i := (xW_g)_i + Z\sigma((xW_{\text{noise}})_i)$, where

1 $Z \sim \mathcal{N}(0, 1)$.

2 σ is the “softplus” activation function, $\sigma(s) = \ln(1 + e^s)$.

2 $\text{KeepTopK}(v, k)_i := \begin{cases} v_i & v_i \text{ is one of the top } k \text{ largest entries in } v \\ -\infty & \text{otherwise.} \end{cases}$

The Gating Mechanism

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

Naive / non-sparse: For trainable weight matrix W_g , compute $G(x) = \text{softmax}(xW_g)$.

Noisy Top-k Gating: Introduces some (tunable) noise and then keeps the top k values:

1 $H(x)_i := (xW_g)_i + Z\sigma((xW_{\text{noise}})_i)$, where

1 $Z \sim \mathcal{N}(0, 1)$.

2 σ is the “softplus” activation function, $\sigma(s) = \ln(1 + e^s)$.

2 $\text{KeepTopK}(v, k)_i := \begin{cases} v_i & v_i \text{ is one of the top } k \text{ largest entries in } v \\ -\infty & \text{otherwise.} \end{cases}$

3 $G(x) = \text{softmax}(\text{KeepTopK}(H(x), k))$.

The Gating Mechanism

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

Naive / non-sparse: For trainable weight matrix W_g , compute $G(x) = \text{softmax}(xW_g)$.

Noisy Top-k Gating: Introduces some (tunable) noise and then keeps the top k values:

1 $H(x)_i := (xW_g)_i + Z\sigma((xW_{\text{noise}})_i)$, where

1 $Z \sim \mathcal{N}(0, 1)$.

2 σ is the “softplus” activation function, $\sigma(s) = \ln(1 + e^s)$.

2 $\text{KeepTopK}(v, k)_i := \begin{cases} v_i & v_i \text{ is one of the top } k \text{ largest entries in } v \\ -\infty & \text{otherwise.} \end{cases}$

3 $G(x) = \text{softmax}(\text{KeepTopK}(H(x), k))$.

Noise is necessary for backpropagation and load balancing, as will be explained later on.

Results: Perplexity

Suppose we have a test set with N tokens, $\{w_1, \dots, w_N\}$.

Let $\mathcal{P}(w_i | w_1, \dots, w_{i-1})$ be the probability that the LLM assigns to the correct i th token, given the preceding $i - 1$ tokens. Intuitively, we want the LLM to assign a high probability to the correct token, so we want the above expression to be as large as possible.

Results: Perplexity

Suppose we have a test set with N tokens, $\{w_1, \dots, w_N\}$.

Let $\mathcal{P}(w_i | w_1, \dots, w_{i-1})$ be the probability that the LLM assigns to the correct i th token, given the preceding $i - 1$ tokens. Intuitively, we want the LLM to assign a high probability to the correct token, so we want the above expression to be as large as possible.

We define

$$\text{perplexity}(\text{LLM}) := \left(\prod_{i=1}^N \mathcal{P}(w_i | w_1, \dots, w_{i-1}) \right)^{-\frac{1}{N}}.$$

Lower perplexity $\Rightarrow$ better performance of the LLM.

Results: Perplexity

Suppose we have a test set with N tokens, $\{w_1, \dots, w_N\}$.

Let $\mathcal{P}(w_i | w_1, \dots, w_{i-1})$ be the probability that the LLM assigns to the correct i th token, given the preceding $i - 1$ tokens. Intuitively, we want the LLM to assign a high probability to the correct token, so we want the above expression to be as large as possible.

We define

$$\text{perplexity}(\text{LLM}) := \left(\prod_{i=1}^N \mathcal{P}(w_i | w_1, \dots, w_{i-1}) \right)^{-\frac{1}{N}}.$$

Lower perplexity $\Rightarrow$ better performance of the LLM.

Perplexity allows us to measure the performance of an LLM independently of the size of the test set.

Results: Perplexity

Suppose we have a test set with N tokens, $\{w_1, \dots, w_N\}$.

Let $\mathcal{P}(w_i | w_1, \dots, w_{i-1})$ be the probability that the LLM assigns to the correct i th token, given the preceding $i-1$ tokens. Intuitively, we want the LLM to assign a high probability to the correct token, so we want the above expression to be as large as possible.

We define

$$\text{perplexity}(\text{LLM}) := \left(\prod_{i=1}^N \mathcal{P}(w_i | w_1, \dots, w_{i-1}) \right)^{-\frac{1}{N}}.$$

Lower perplexity $\Rightarrow$ better performance of the LLM.

Perplexity allows us to measure the performance of an LLM independently of the size of the test set.

Model	Parameters	Test Perplexity
Best prior baseline	151M	30.6
MoE (4096 experts)	4B	28.0

Table: Performance on the 1 Billion Word Language Modeling Benchmark. The MoE model has 8.5% lower perplexity, while using fewer ops/timestep (142.7M vs. 151M) and training faster (47 hrs vs. 59 hrs, both on 32 K40 GPUs). Data from [SMM].

Results: BLEU

BLEU (Bilingual Evaluation Understudy) is the standard metric for scoring machine translation quality.

Results: BLEU

BLEU (Bilingual Evaluation Understudy) is the standard metric for scoring machine translation quality.

Define

1 p_n = the fraction of the LLM's n -grams that also occur in the reference.

$$2 \quad BP = \begin{cases} 1 & \text{LLM'S output is at least as long as the reference,} \\ e^{1-r/c} & \text{otherwise.} \end{cases}$$

Here, c is the length of the LLM's output and r is the length of the reference.

Results: BLEU

BLEU (Bilingual Evaluation Understudy) is the standard metric for scoring machine translation quality.

Define

1 p_n = the fraction of the LLM's n -grams that also occur in the reference.

2 $BP = \begin{cases} 1 & \text{LLM'S output is at least as long as the reference,} \\ e^{1-r/c} & \text{otherwise.} \end{cases}$

Here, c is the length of the LLM's output and r is the length of the reference.

Formally,

$$\text{BLEU(LLM)} := BP \cdot \exp \left(\frac{1}{4} \sum_{n=1}^4 \ln(p_n) \right).$$

Higher BLEU => better performance of the LLM.

Results: BLEU cont.

Higher BLEU => better performance of the LLM.

Task	GNMT baseline	MoE	Δ
WMT'14 En→Fr	39.22	40.56	+1.34
WMT'14 En→De	24.91	26.03	+1.12
Google Prod. En→Fr	35.56	36.57	+1.01

Table: Performance on several translation tasks. On the production dataset, the MoE model also trained 6x faster (1 day on 64 K40s vs. 6 days on 96 K80s). Data from [SMM].

Load Balancing

Load Balancing

One big issue: Suppose each expert runs on a separate device.

- 1 After a few training iterations, the gating network converges to mostly activate the same few experts => self-reinforces, as favored experts are trained quicker and hence selected more.

Load Balancing

One big issue: Suppose each expert runs on a separate device.

- 1 After a few training iterations, the gating network converges to mostly activate the same few experts => self-reinforces, as favored experts are trained quicker and hence selected more.
- 2 Imagine that ten tokens come in, but instead of splitting evenly, five land on one expert, while the other five scatter across five different experts. These small, irregular batches are inefficient to run on hardware.

Load Balancing

One big issue: Suppose each expert runs on a separate device.

- 1 After a few training iterations, the gating network converges to mostly activate the same few experts => self-reinforces, as favored experts are trained quicker and hence selected more.
- 2 Imagine that ten tokens come in, but instead of splitting evenly, five land on one expert, while the other five scatter across five different experts. These small, irregular batches are inefficient to run on hardware.

Shazeer et al. avoid this by adding an **auxiliary loss** to encourage giving all experts equal importance => all experts receive training tokens.

Auxiliary Loss Part 1: Importance

The auxiliary loss is comprised of two losses working together to encourage balanced expert utilization.

We consider N tokens, each with dimension d . Let X be an $N \times d$ matrix, where the i th row corresponds to the i th token embedding.

Auxiliary Loss Part 1: Importance

The auxiliary loss is comprised of two losses working together to encourage balanced expert utilization.

We consider N tokens, each with dimension d . Let X be an $N \times d$ matrix, where the i th row corresponds to the i th token embedding.

Importance: Recall $y = \sum_{i=1}^n G(x)_i E_i(x)$. Define $\text{Importance}(X) := \sum_{x \in X} G(x) \in \mathbb{R}^n$, the total gate weight each expert receives over batch X .

$$\mathcal{L}_{\text{importance}}(X) := w_{\text{importance}} \cdot \text{CV}(\text{Importance}(X))^2,$$

where $\text{CV}(v) = \text{std}(v)/\text{mean}(v)$ is the coefficient of variation and $w_{\text{importance}}$ is a hand-tuned constant.

Intuition: As the variation in popularity of experts increases (i.e. some experts get VERY popular), $\mathcal{L}_{\text{importance}}(X)$ increases.

Auxiliary Loss Part 2: Load

Load: $\text{Load}(X)_i$ equals the number of tokens routed to expert i . Without adding noise to the gating mechanism, this is a discrete count $\Rightarrow$ not differentiable, so we can't back-propagate through it directly.

Auxiliary Loss Part 2: Load

Load: $\text{Load}(X)_i$ equals the number of tokens routed to expert i . Without adding noise to the gating mechanism, this is a discrete count $\Rightarrow$ not differentiable, so we can't back-propagate through it directly.

The solution is the noise function added in noisy top-k gating. Recall

$$G(x) = \text{softmax}(\text{KeepTopK}(H(x), k)),$$

where $H(x)_i := (xW_g)_i + Z\sigma((xW_{\text{noise}})_i)$.

Auxiliary Loss Part 2: Load

Load: $\text{Load}(X)_i$ equals the number of tokens routed to expert i . Without adding noise to the gating mechanism, this is a discrete count $\Rightarrow$ not differentiable, so we can't back-propagate through it directly.

The solution is the noise function added in noisy top-k gating. Recall

$$G(x) = \text{softmax}(\text{KeepTopK}(H(x), k)),$$

where $H(x)_i := (xW_g)_i + Z\sigma((xW_{\text{noise}})_i)$.

Let $T_i(x)$ equal the k th largest entry of $H(x)$ if we replaced $H(x)_i$ with $-\infty$. Intuitively, $T_i(x)$ is the element $H(x)_i$ needs to beat if it makes it into the top k .

Auxiliary Loss Part 2: Load

Load: $\text{Load}(X)_i$ equals the number of tokens routed to expert i . Without adding noise to the gating mechanism, this is a discrete count $\Rightarrow$ not differentiable, so we can't back-propagate through it directly.

The solution is the noise function added in noisy top-k gating. Recall

$$G(x) = \text{softmax}(\text{KeepTopK}(H(x), k)),$$

where $H(x)_i := (xW_g)_i + Z\sigma((xW_{\text{noise}})_i)$.

Let $T_i(x)$ equal the k th largest entry of $H(x)$ if we replaced $H(x)_i$ with $-\infty$. Intuitively, $T_i(x)$ is the element $H(x)_i$ needs to beat if it makes it into the top k .

If $P(x, i)$ is the probability that the i th expert is selected in noisy top-k gating,

$$P(x, i) = \mathcal{P}(H(x)_i > T_i(x)) = \mathcal{P}\left(Z > \frac{(xW_g)_i - T_i(x)}{\sigma((xW_{\text{noise}})_i)}\right).$$

Auxiliary Loss Part 2: Load cont.

Load: $\text{Load}(X)_i$ equals the number of tokens routed to expert i .

If $P(x, i)$ is the probability that the i th expert is selected in noisy top-k gating,

$$P(x, i) = \mathcal{P}(H(x)_i > T_i(x)) = \mathcal{P}\left(Z > \frac{(xW_g)_i - T_i(x)}{\sigma((xW_{\text{noise}})_i)}\right).$$

Auxiliary Loss Part 2: Load cont.

Load: $\text{Load}(X)_i$ equals the number of tokens routed to expert i .

If $P(x, i)$ is the probability that the i th expert is selected in noisy top-k gating,

$$P(x, i) = \mathcal{P}(H(x)_i > T_i(x)) = \mathcal{P}\left(Z > \frac{(xW_g)_i - T_i(x)}{\sigma((xW_{\text{noise}})_i)}\right).$$

We can take the derivative of the above expression!

$$\frac{\partial P(x, i)}{\partial (xW_g)_i} = \frac{1}{\sigma((xW_{\text{noise}})_i)} \varphi\left(\frac{(xW_g)_i - T_i(x)}{\sigma((xW_{\text{noise}})_i)}\right),$$

where φ is the standard normal Gaussian.

Auxiliary Loss Part 2: Load cont.

Load: $\text{Load}(X)_i$ equals the number of tokens routed to expert i .

If $P(x, i)$ is the probability that the i th expert is selected in noisy top-k gating,

$$P(x, i) = \mathcal{P}(H(x)_i > T_i(x)) = \mathcal{P}\left(Z > \frac{(xW_g)_i - T_i(x)}{\sigma((xW_{\text{noise}})_i)}\right).$$

We can take the derivative of the above expression!

$$\frac{\partial P(x, i)}{\partial (xW_g)_i} = \frac{1}{\sigma((xW_{\text{noise}})_i)} \varphi\left(\frac{(xW_g)_i - T_i(x)}{\sigma((xW_{\text{noise}})_i)}\right),$$

where φ is the standard normal Gaussian.

As a result, $\mathcal{L}_{\text{load}}(X) := w_{\text{load}} \cdot CV(\text{Load}(X))^2$ is trainable by gradient descent. Here, w_{load} is a hand-tuned constant.

Expert Capacity

Two sources of imbalance:

- 1 Only a few experts are chosen. Solution: Auxiliary loss [SMM].
- 2 Some experts get a disproportionately large share of tokens. Solution: Expert capacity [LLX].

Expert Capacity

Two sources of imbalance:

- 1 Only a few experts are chosen. Solution: Auxiliary loss [SMM].
- 2 Some experts get a disproportionately large share of tokens. Solution: Expert capacity [LLX].

Expert Capacity: Limit the number of tokens each expert can receive. If an expert has already received a large number of tokens, all remaining tokens assigned to it are instead passed to the next layer.

Expert Capacity

Two sources of imbalance:

- 1 Only a few experts are chosen. Solution: Auxiliary loss [SMM].
- 2 Some experts get a disproportionally large share of tokens. Solution: Expert capacity [LLX].

Expert Capacity: Limit the number of tokens each expert can receive. If an expert has already received a large number of tokens, all remaining tokens assigned to it are instead passed to the next layer.

New source of nonsmoothness => motivated future work [LBD, ZLL].

The Capacity Factor

The capacity factor equation

- 1 **Expert capacity**: How many tokens each expert can process.
- 2 **Capacity factor**: A hyperparameter that determines how many tokens an expert can process.

$$\text{expert capacity} = \frac{\text{tokens per batch}}{\text{number of experts}} \times \text{capacity factor}.$$

Based on the calculated expert capacity, we allocate memory / computing power / network bandwidth to each expert.

The Capacity Factor

The capacity factor equation

- 1 **Expert capacity**: How many tokens each expert can process.
- 2 **Capacity factor**: A hyperparameter that determines how many tokens an expert can process.

$$\text{expert capacity} = \frac{\text{tokens per batch}}{\text{number of experts}} \times \text{capacity factor}.$$

Based on the calculated expert capacity, we allocate memory / computing power / network bandwidth to each expert.

Example: Suppose we have six tokens and three experts.

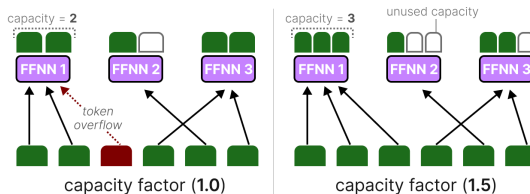


Figure: Setting the capacity factor equal to 1.0, each expert gets an appropriate number of tokens. Setting the capacity to 1.5, some of the experts do not receive enough tokens, leading to a waste of allocated resources. Figure from [GRO].

The Capacity Factor: A Probabilistic View

Idealization: Each of the N tokens in a batch is routed independently and uniformly at random to one of n experts.

The Capacity Factor: A Probabilistic View

Idealization: Each of the N tokens in a batch is routed independently and uniformly at random to one of n experts.

Let X_i equal the number of tokens routed to expert i . Since each token independently lands on expert i with probability $1/n$,

$$X_i \sim \text{Binomial}(N, 1/n), \quad \mu := \mathbb{E}[X_i] = \frac{N}{n}.$$

$\mu = N/n$ is exactly the “tokens per batch / number of experts” term in the capacity factor equation. A capacity factor of 1.0 sets expert capacity at exactly the mean load.

The Capacity Factor: A Probabilistic View

Idealization: Each of the N tokens in a batch is routed independently and uniformly at random to one of n experts.

Let X_i equal the number of tokens routed to expert i . Since each token independently lands on expert i with probability $1/n$,

$$X_i \sim \text{Binomial}(N, 1/n), \quad \mu := \mathbb{E}[X_i] = \frac{N}{n}.$$

$\mu = N/n$ is exactly the “tokens per batch / number of experts” term in the capacity factor equation. A capacity factor of 1.0 sets expert capacity at exactly the mean load.

How far above the mean should we expect the busiest expert to be? That tells us how much slack the capacity factor actually needs to provide.

A Concentration Bound

Fix $\delta \in (0, 1]$ and let $1 + \delta$ be the capacity factor:

$$\text{expert capacity} = \frac{N}{n} \times (1 + \delta) = (1 + \delta)\mu.$$

A Concentration Bound

Fix $\delta \in (0, 1]$ and let $1 + \delta$ be the capacity factor:

$$\text{expert capacity} = \frac{N}{n} \times (1 + \delta) = (1 + \delta)\mu.$$

A standard (simplified) multiplicative Chernoff bound: for $0 < \delta \leq 1$,

$$\mathcal{P}(X_i \geq (1 + \delta)\mu) \leq e^{-\mu\delta^2/3}.$$

A Concentration Bound

Fix $\delta \in (0, 1]$ and let $1 + \delta$ be the capacity factor:

$$\text{expert capacity} = \frac{N}{n} \times (1 + \delta) = (1 + \delta)\mu.$$

A standard (simplified) multiplicative Chernoff bound: for $0 < \delta \leq 1$,

$$\mathcal{P}(X_i \geq (1 + \delta)\mu) \leq e^{-\mu\delta^2/3}.$$

Union bound over all n experts:

$$\mathcal{P}\left(\max_i X_i \geq (1 + \delta)\mu\right) \leq ne^{-\mu\delta^2/3}.$$

A Concentration Bound

Fix $\delta \in (0, 1]$ and let $1 + \delta$ be the capacity factor:

$$\text{expert capacity} = \frac{N}{n} \times (1 + \delta) = (1 + \delta)\mu.$$

A standard (simplified) multiplicative Chernoff bound: for $0 < \delta \leq 1$,

$$\mathcal{P}(X_i \geq (1 + \delta)\mu) \leq e^{-\mu\delta^2/3}.$$

Union bound over all n experts:

$$\mathcal{P}\left(\max_i X_i \geq (1 + \delta)\mu\right) \leq ne^{-\mu\delta^2/3}.$$

We want this failure probability to vanish as n grows – set it $\leq 1/n$ and solve for δ :

$$ne^{-\mu\delta^2/3} \leq \frac{1}{n} \implies \delta \geq \sqrt{\frac{6 \ln n}{\mu}} = \sqrt{\frac{6n \ln n}{N}}.$$

A Concentration Bound

Fix $\delta \in (0, 1]$ and let $1 + \delta$ be the capacity factor:

$$\text{expert capacity} = \frac{N}{n} \times (1 + \delta) = (1 + \delta)\mu.$$

A standard (simplified) multiplicative Chernoff bound: for $0 < \delta \leq 1$,

$$\mathcal{P}(X_i \geq (1 + \delta)\mu) \leq e^{-\mu\delta^2/3}.$$

Union bound over all n experts:

$$\mathcal{P}\left(\max_i X_i \geq (1 + \delta)\mu\right) \leq ne^{-\mu\delta^2/3}.$$

We want this failure probability to vanish as n grows – set it $\leq 1/n$ and solve for δ :

$$ne^{-\mu\delta^2/3} \leq \frac{1}{n} \implies \delta \geq \sqrt{\frac{6 \ln n}{\mu}} = \sqrt{\frac{6n \ln n}{N}}.$$

In sum, define $\delta = \sqrt{\frac{6n \ln n}{N}}$. If we set the capacity factor equal to $1 + \delta$, then with probability at least $1 - 1/n$, no expert's load exceeds the expert capacity $(1 + \delta)\mu$.

Switch Routing and Switch Transformers

Switch Routing

“Shazeer et al. conjectured that routing to $k > 1$ experts was necessary... we instead use a simplified strategy where we route to only a single expert. We show this simplification preserves model quality, reduces routing computation and performs better” [FZS].

Switch Routing

“Shazeer et al. conjectured that routing to $k > 1$ experts was necessary... we instead use a simplified strategy where we route to only a single expert. We show this simplification preserves model quality, reduces routing computation and performs better” [FZS].

Fedus et al. introduce **switch routing**: $p(x) = \text{softmax}(xW_g)$,

$$G(x)_i = \begin{cases} p_i(x) & p_i(x) = \max_{1 \leq j \leq n} p_j(x) \\ 0 & \text{otherwise.} \end{cases}$$

Switch Routing

“Shazeer et al. conjectured that routing to $k > 1$ experts was necessary... we instead use a simplified strategy where we route to only a single expert. We show this simplification preserves model quality, reduces routing computation and performs better” [FZS].

Fedus et al. introduce **switch routing**: $p(x) = \text{softmax}(xW_g)$,

$$G(x)_i = \begin{cases} p_i(x) & p_i(x) = \max_{1 \leq j \leq n} p_j(x) \\ 0 & \text{otherwise.} \end{cases}$$

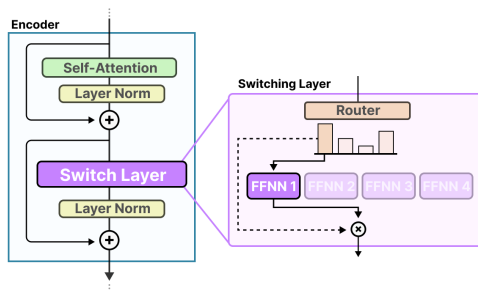


Figure: The Switch Transformer architecture. Figure from [GRO].

The Switch Transformer Loss

Recall: $p(x) = \text{softmax}(xW_g)$, and $G(x)_i = p_i(x)$ if expert i wins, else 0.

The Switch Transformer Loss

Recall: $p(x) = \text{softmax}(xW_g)$, and $G(x)_i = p_i(x)$ if expert i wins, else 0.

For a batch X of N tokens, define two quantities per expert:

$$f_i := \frac{1}{N} \sum_{x \in X} \mathbb{1} \left[i = \arg \max_{1 \leq j \leq n} p_j(x) \right] \quad (\text{fraction of tokens routed to expert } i),$$

$$P_i := \frac{1}{N} \sum_{x \in X} p_i(x) \quad (\text{average router probability on expert } i).$$

The Switch Transformer Loss

Recall: $p(x) = \text{softmax}(xW_g)$, and $G(x)_i = p_i(x)$ if expert i wins, else 0.

For a batch X of N tokens, define two quantities per expert:

$$f_i := \frac{1}{N} \sum_{x \in X} \mathbb{1} \left[i = \arg \max_{1 \leq j \leq n} p_j(x) \right] \quad (\text{fraction of tokens routed to expert } i),$$

$$P_i := \frac{1}{N} \sum_{x \in X} p_i(x) \quad (\text{average router probability on expert } i).$$

The auxiliary loss is

$$\mathcal{L}_{\text{aux}}(X) := \alpha n \sum_{i=1}^n f_i P_i, \quad \alpha \text{ a hand-tuned constant.}$$

The Switch Transformer Loss

Recall: $p(x) = \text{softmax}(xW_g)$, and $G(x)_i = p_i(x)$ if expert i wins, else 0.

For a batch X of N tokens, define two quantities per expert:

$$f_i := \frac{1}{N} \sum_{x \in X} \mathbb{1} \left[i = \arg \max_{1 \leq j \leq n} p_j(x) \right] \quad (\text{fraction of tokens routed to expert } i),$$

$$P_i := \frac{1}{N} \sum_{x \in X} p_i(x) \quad (\text{average router probability on expert } i).$$

The auxiliary loss is

$$\mathcal{L}_{\text{aux}}(X) := \alpha n \sum_{i=1}^n f_i P_i, \quad \alpha \text{ a hand-tuned constant.}$$

Since $\sum_i f_i = \sum_i P_i = 1$, this is small exactly when both vectors are close to uniform, no single expert is simultaneously overused (f_i large) and strongly favored (P_i large) at once.

Why Backpropagation Doesn't Break

f_i is built from an indicator/argmax, so it's piecewise constant in W_g : its gradient is 0 almost everywhere.

Why Backpropagation Doesn't Break

f_i is built from an indicator/argmax, so it's piecewise constant in W_g : its gradient is 0 almost everywhere.

P_i is just an average of softmax outputs, smooth everywhere. Treating f_i as a constant,

$$\nabla_{W_g} \mathcal{L}_{\text{aux}} = \alpha n \sum_{i=1}^n f_i \cdot \nabla_{W_g} P_i,$$

a perfectly ordinary, closed-form gradient. All of $\mathcal{L}_{\text{aux}}$'s gradient signal flows through P_i .

Why Backpropagation Doesn't Break

f_i is built from an indicator/argmax, so it's piecewise constant in W_g : its gradient is 0 almost everywhere.

P_i is just an average of softmax outputs, smooth everywhere. Treating f_i as a constant,

$$\nabla_{W_g} \mathcal{L}_{\text{aux}} = \alpha n \sum_{i=1}^n f_i \cdot \nabla_{W_g} P_i,$$

a perfectly ordinary, closed-form gradient. All of $\mathcal{L}_{\text{aux}}$'s gradient signal flows through P_i .

Separately, the main task loss flows through $y = p_{i^*}(x)E_{i^*}(x)$. Here, $p_{i^*}(x)$ is the winning expert's softmax value, smooth in W_g away from the (measure-zero) boundary where the arg max switches.

Results: Quality

[FZS] compare the performance of several LLMs against the corresponding “switch” version. Each of the rows in the table below corresponds to a different tasks and each column corresponds to a different LLM. For each row, higher scores are better.

Task	T5-Base	Switch-Base	T5-Large	Switch-Large
GLUE	84.3	86.7	87.8	88.5
SuperGLUE	75.1	79.5	82.7	84.7
SQuAD	85.5	87.2	88.1	88.6
Winogrande	66.6	73.3	79.1	83.0

Table: Sparsity isn't just faster training; it improves quality at equal compute. For example, with respect to the Super-GLUE task, the switch architecture increased the score by +4.4 compared to the original BASE model, and increased the score by +2.0 compared to the original Large model.

Across 101 languages, Switch Transformer improves perplexity over the dense mT5 baseline on every single one, with a mean speedup of 5x. 91% of languages reached at least a 4x speedup.

Conclusion

Conclusion

Mixture of Expert layers contains a certain number of experts, where each expert is a neural network.

Sparsity means that for each token, only a few experts are used => models with more parameters at a comparable computational cost.

Load balancing is the problem of ensuring that all experts receive a similar number of tokens. Researchers introduce **auxiliary loss** and **expert capacity** for improved load balancing.

Switch Transformers replace the standard FFN with a MoE layer, leading to enhanced performance at equal compute.

References

- FZS** Fedus, William, Barret Zoph, and Noam Shazeer. “Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity,” 2021. 2101.03961.
- GRO** Grootendorst, Maarten. “A Visual Guide to Mixture of Experts (MoE),” 2024. [Link](#).
- JJN** Jacobs, Robert, Michael Jordan, Steven Nowlan, and Geoffrey Hinton. “Adaptive Mixtures of Local Experts,” 1991. 10.1162/neco.1991.3.1.79.
- JM** Jurafsky, Daniel and James H. Martin. “Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models,” 3rd edition draft, 2026.
- LBD** Lewis, Mike, Shruti Bhosale, Tim Dettmers, Naman Goyal, Luke Zettlemoyer. “BASE Layers: Simplifying Training of Large, Sparse Models,” 2021. 2103.16716.

References

- LLX** Lepikhin, Dmitry, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, Zhifeng Chen. "GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding," 2020. 2006.16668.
- SMM** Shazeer, Noam, Azalia Mirhoseini, Krzysztof Maziarsz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," 2017. 1701.06538.
- STS** Sanseviero, Omar, Lewis Tunstall, Philipp Schmid, Sourab Mangrulkar, Younes B, and Pedro Cuenca. "Mixture of Experts Explained," 2023. [Link](#).
- ZLL** Zhou, Yanqi, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew Dai, Zhifeng Chen, Quoc Le, James Laudon. "Mixture-of-Experts with Expert Choice Routing," 2022. 2202.09368.