

# Dissecting a Transformer: Other Components

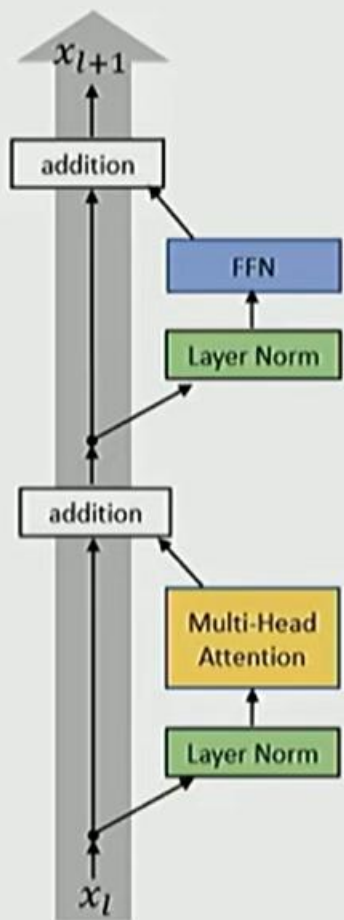
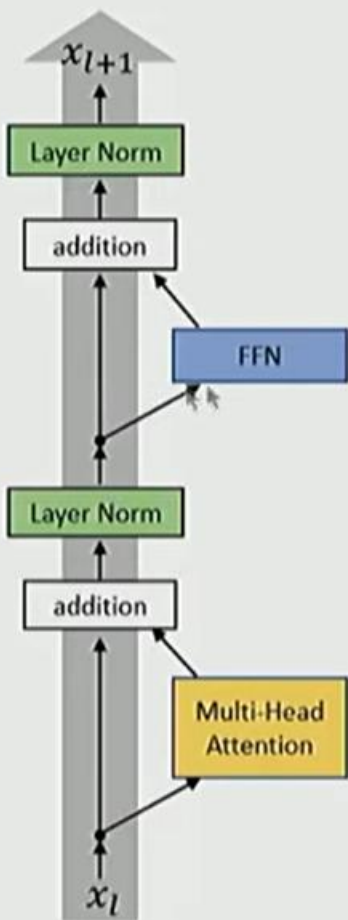
Mausam

(Based on slides of Tatsunori Hashimoto, Jaisidh Singh)

# Other Variations

- Normalizations
- Activations, FFN
- Hyperparameters that matter/don't matter
- Stability Tricks

Aa Name	#	Year	Norm	Parallel Layer	Pre-norm	Position embedding	Activations
Original transformer		2017	LayerNorm	Serial	<input type="checkbox"/>	Sine	ReLU
GPT		2018	LayerNorm	Serial	<input type="checkbox"/>	Absolute	GeLU
T5 (11B)		2019	RMSNorm	Serial	<input checked="" type="checkbox"/>	Relative	ReLU
GPT2		2019	LayerNorm	Serial	<input checked="" type="checkbox"/>	Sine	GeLU
T5 (XXL 11B) v1.1		2020	RMSNorm	Serial	<input checked="" type="checkbox"/>	Relative	GeGLU
mT5		2020	RMSNorm	Serial	<input checked="" type="checkbox"/>	Relative	GeGLU
GPT3 (175B)		2020	LayerNorm	Serial	<input checked="" type="checkbox"/>	Sine	GeLU
GPTJ		2021	LayerNorm	Parallel	<input checked="" type="checkbox"/>	RoPE	GeLU
LaMDA		2021			<input checked="" type="checkbox"/>	Relative	GeGLU
Gopher (280B)		2021	RMSNorm	Serial	<input checked="" type="checkbox"/>	Relative	ReLU
GPT-NeoX		2022	LayerNorm	Parallel	<input checked="" type="checkbox"/>	RoPE	GeLU
BLOOM (175B)		2022	LayerNorm	Serial	<input checked="" type="checkbox"/>	Alibi	GeLU
OPT (175B)		2022	LayerNorm	Serial	<input checked="" type="checkbox"/>	Absolute	ReLU
PaLM (540B)		2022	RMSNorm	Parallel	<input checked="" type="checkbox"/>	RoPE	SwiGLU
Chinchilla		2022	RMSNorm	Serial	<input checked="" type="checkbox"/>	Relative	ReLU
Mistral (7B)		2023	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
LLaMA2 (70B)		2023	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
LLaMA (65B)		2023	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
Qwen (14B)		2024	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
DeepSeek (67B)		2024	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
Yi (34B)		2024	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU

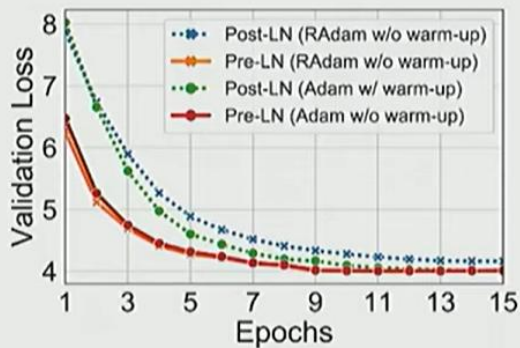


## Warmup in Post-LN

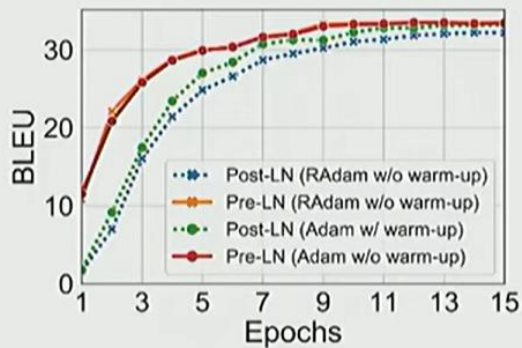
- learning rate starts at a very low value (often zero or near-zero) and increases linearly or exponentially to the target maximum learning rate over the first few thousand iterations or percentage of total training steps
- Needed to control early gradients, calculate momentum more accurately, prevent exploding gradients

# Pre-LN vs Post-LN

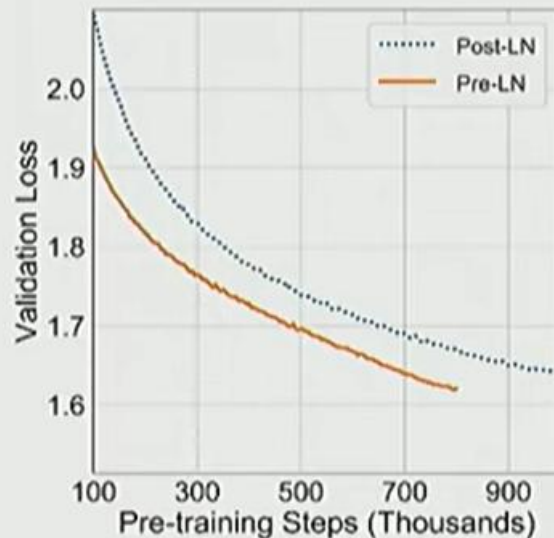
- Pre-LN: Reduces the need for warmup
- Well-behaved gradients  $\rightarrow$  better training



(a) Validation Loss (IWSLT)



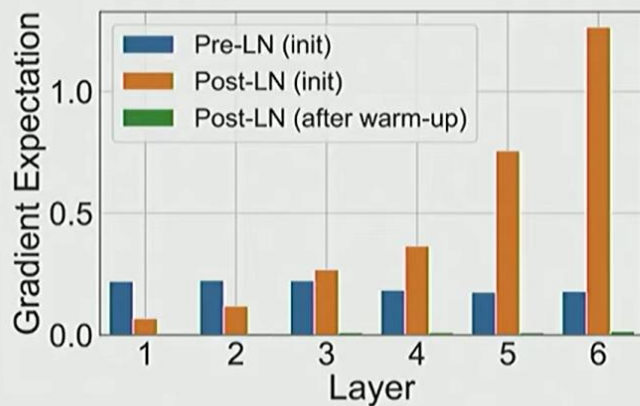
(b) BLEU (IWSLT)



(a) Validation Loss on BERT

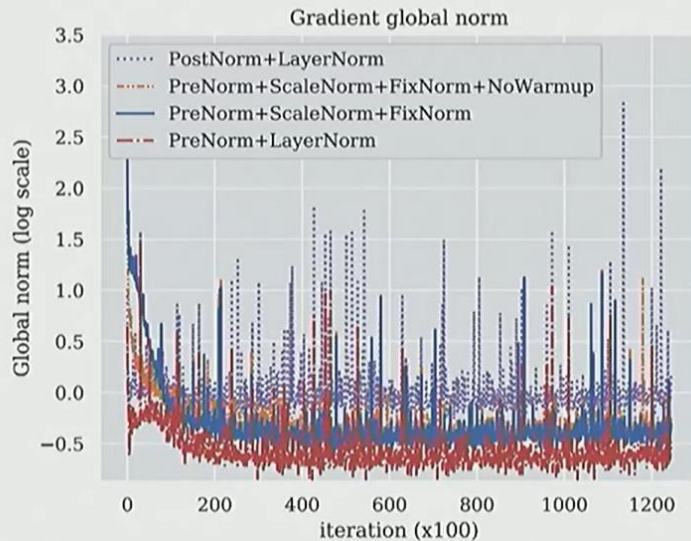
# Pre-LN vs Post-LN

Gradient attenuation [Xiong 2020]



(a)  $W^1$  in the FFN sub-layers

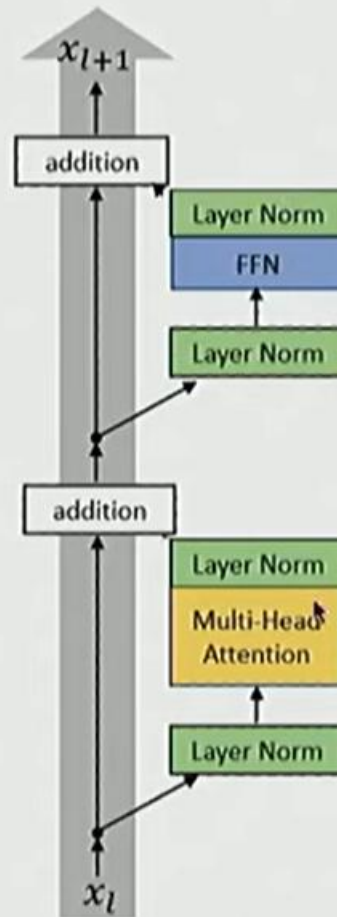
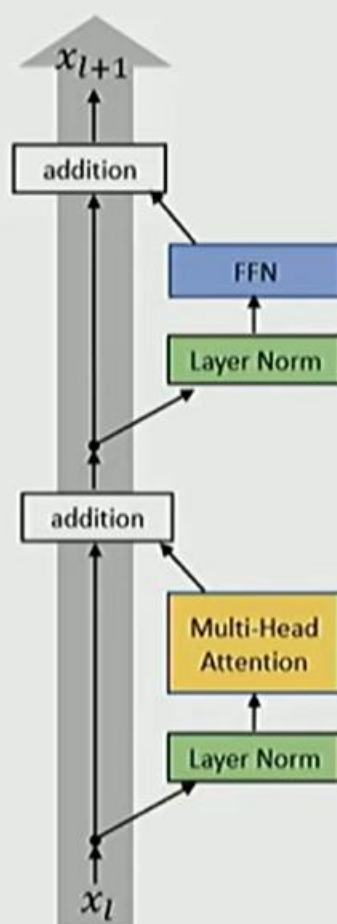
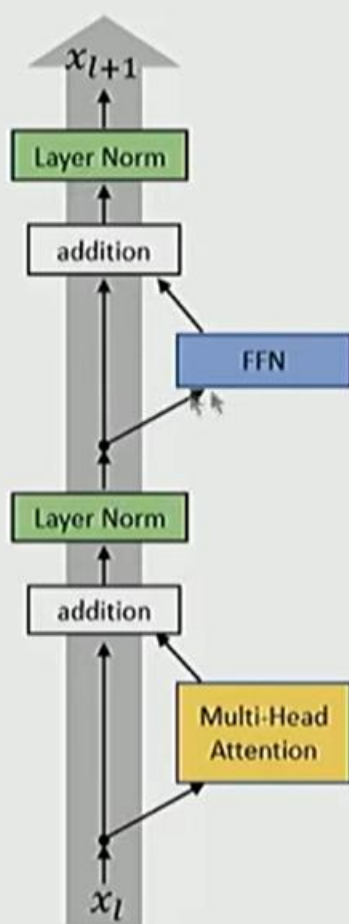
Gradient spikes [Salazar and Nguyen]



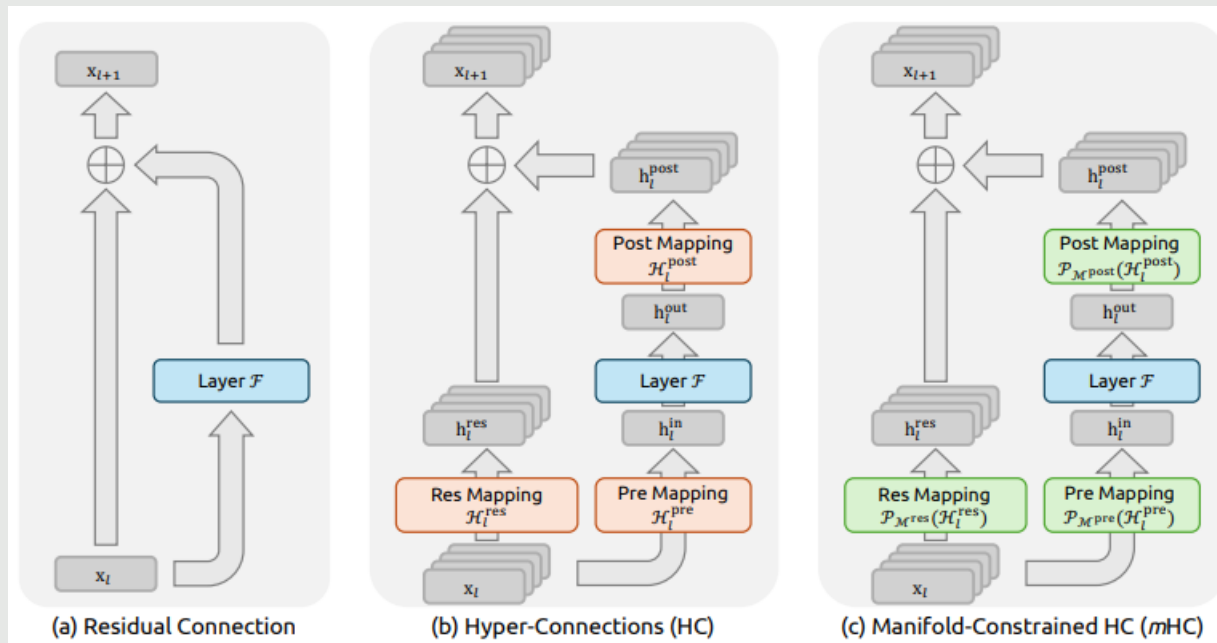
- Original stated benefit: removing warmup
- 6 • Today: stability and therefore larger LR for large networks

# Sandwich Norm

- Double norm/Peri-LN
  - Ensures both inputs and outputs are well behaved
  - Gradients even more well-behaved
  - Stabler training



# Manifold-Constrained HyperConnections



- mHC = increase expressiveness of residual stream but
- constrain the mixing matrices to a manifold so signal propagation remains stable.

# Form of Normalization

- Once Pre-LN became standard practice, practitioners realized that models needed more scale control, less centering.

Original transformer: **LayerNorm** – normalizes the mean and variance across  $d_{model}$

$$y = \frac{x - \mathbf{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

Many modern LMs: **RMSNorm** – does not subtract mean or add a bias term

$$y = \frac{x}{\sqrt{\|x\|_2^2 + \epsilon}} * \gamma$$

# Why RMSNorm?

- Modern Explanation
  - Fewer operations (no mean)
  - Fewer parameters (no bias term to store)
- Does this make sense?

Operator class	% flop
△ Tensor contraction	99.80
□ Stat. normalization	0.17
○ Element-wise	0.03

Operator class	% flop	% Runtime
△ Tensor contraction	99.80	61.0
□ Stat. normalization	0.17	25.5
○ Element-wise	0.03	13.5

# Expts: RMSNorm vs LayerNorm

- Narang et al 2020

Model	Params	Ops	Step/s	Early loss	Final loss	SGLUE	XSum	WebQ	WMT EnDe
Vanilla Transformer	223M	11.1T	3.50	2.182 ± 0.005	1.838	71.66	17.78	23.02	26.62
RMS Norm	223M	11.1T	3.68	2.167 ± 0.008	<b>1.821</b>	<b>75.45</b>	<b>17.94</b>	<b>24.07</b>	<b>27.14</b>
Rezero	223M	11.1T	3.51	2.262 ± 0.003	1.939	61.69	15.64	20.90	26.37
Rezero + LayerNorm	223M	11.1T	3.26	2.223 ± 0.006	1.858	70.42	17.58	23.02	26.29
Rezero + RMS Norm	223M	11.1T	3.34	2.221 ± 0.009	1.875	70.33	17.32	23.02	26.19
Fixup	223M	11.1T	2.95	2.382 ± 0.012	2.067	58.56	14.42	23.02	26.31

# Bias Terms

**Most modern transformers**

Original Transformer:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Most implementations (if they're not gated):

$$\text{FFN}(x) = \sigma(xW_1)W_2$$

Bias allows a neuron to say:

“Activate even if input is zero.”

But in deep residual networks with normalization, this is unnecessary because:

- the residual stream already carries signal
- normalization centers activations

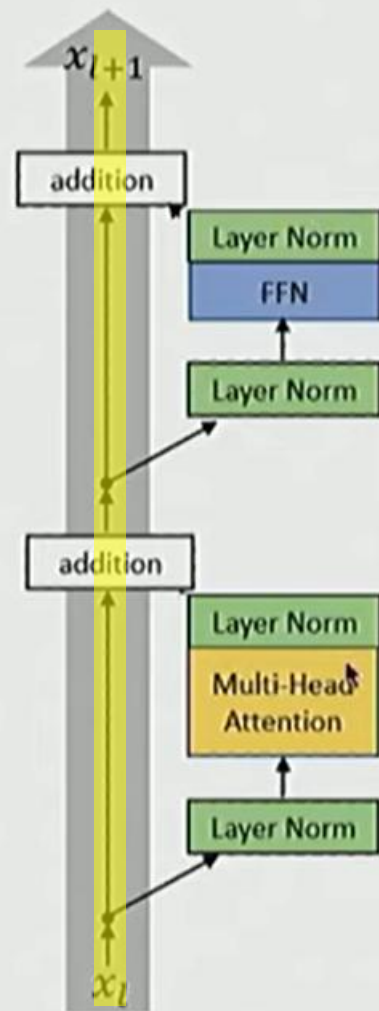
So bias becomes **extra freedom without benefit**, which can hurt stability.

# Attention Residuals

- Residual Learning ( $h = x$  in figure)

$$h_l = h_{l-1} + f_{l-1}(h_{l-1})$$

- Issues
  - Hidden states grow in magnitude as depth increases*
  - Early-layer information gets diluted*
  - Later layers must “shout louder” (produce larger outputs) to matter*
  - The model mixes everything without control*



# Previous Solution

- Highway Networks

$$\mathbf{h}_l = (1 - \mathbf{g}_l) \odot \mathbf{h}_{l-1} + \mathbf{g}_l \odot f_{l-1}(\mathbf{h}_{l-1})$$

- Key Issue
  - Information lost during aggregation cannot be recovered in deeper layers
- Solution?

# Proposed Solution

- Attention Residuals (Full)

$$h_l = \alpha_{0 \rightarrow l} \cdot h_1 + \sum_{i=1}^{l-1} \alpha_{i \rightarrow l} \cdot f_i(h_i)$$

- Some layers get high importance*
  - Some layers are almost ignored*
  - The selection changes dynamically per input*
- Issues
    - Need for storage during inference
    - Need for additional communication during training (e.g., layers split across GPUs)

$$\alpha_{i \rightarrow l} = \frac{\phi(q_l, k_i)}{\sum_{j=0}^{l-1} \phi(q_l, k_j)}$$

$$q_l = w_l, \quad k_i = v_i = \begin{cases} h_1 & i = 0 \\ f_i(h_i) & 1 \leq i \leq l-1 \end{cases}$$

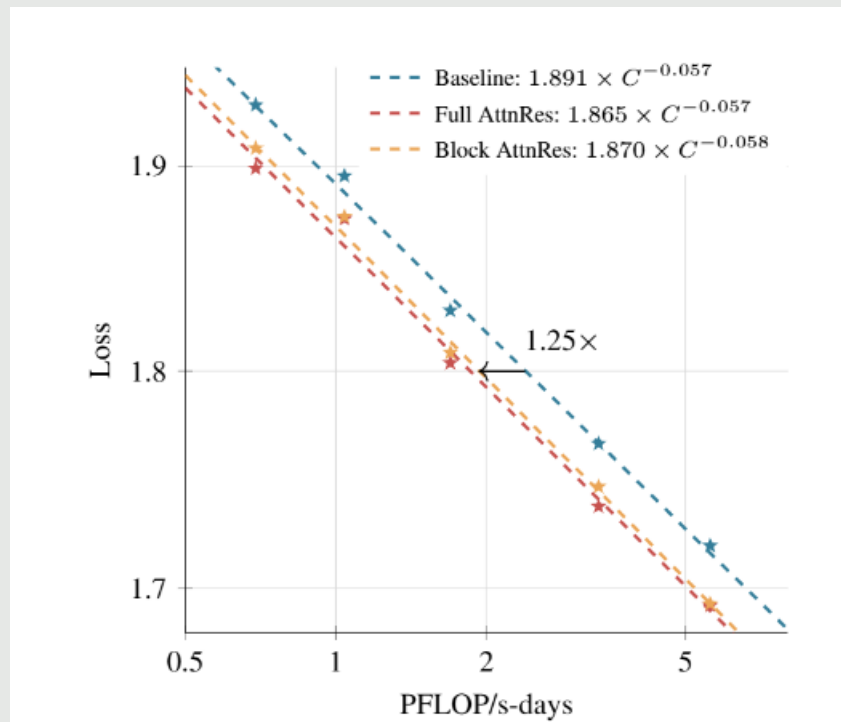
# Block Attention Residuals

- Split layers into blocks
- Intra-block residual
  - Standard residual stream within a block
- Inter-block attention

$$\mathbf{b}_n = \sum_{j \in \mathcal{B}_n} f_j(\mathbf{h}_j)$$

$$\mathbf{V} = \begin{cases} [\mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_{n-1}]^\top & \text{if } i = 1 \text{ (first layer of block } n) \\ [\mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_{n-1}, \mathbf{b}_n^{i-1}]^\top & \text{if } i \geq 2 \text{ (subsequent layers)} \end{cases}$$

# Results



# Normalization: Summary

- Basically everyone does pre-norm.
  - Intuition – keep the good parts of residual connections
  - Observations – nicer gradient propagation, fewer spike
  - Some people add a second norm outside the residual stream (NOT post-norm)
- Most people do RMSnorm
  - In practice, works as well as LayerNorm
  - But, has fewer parameters to move around, which saves on wallclock time
  - People more generally drop bias terms since the compute/param tradeoffs are not great.

# Other Variations

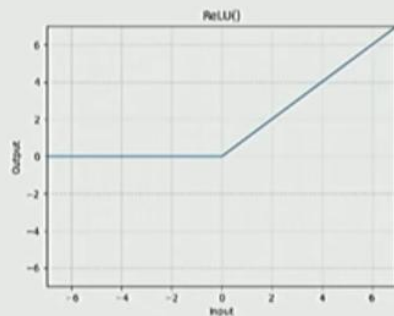
- Normalizations
- Activations, FFN
- Hyperparameters that matter/don't matter
- Stability Tricks

# Activations

- A large set of activations
  - ReLU, GeLU, Swish, ELU, GLU, GeGLU, ReGLU, SeLU, SwiGLU, LiGLU...
- Which ones do people use? Do they matter?

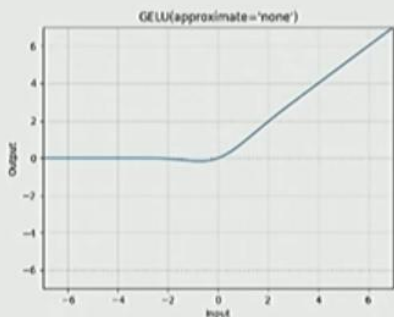
## ReLU

$$FF(x) = \max(0, xW_1) W_2$$



## GeLU

$$FF(x) = \text{GELU}(xW_1)W_2$$
$$\text{GELU}(x) := x\Phi(x)$$



# Gated Linear Units

GLUs modify the ‘first part’ of a FF layer

$$FF(x) = \max(0, xW_1) W_2$$

Instead of a linear + ReLU, augment the above with an (entrywise) linear term

$$\max(0, xW_1) \rightarrow \max(0, xW_1) \otimes (xV)$$

This gives the gated variant (ReGLU) – note that we have an extra parameter (V)

$$FF_{\text{ReGLU}}(x) = (\max(0, xW_1) \otimes xV) W_2$$

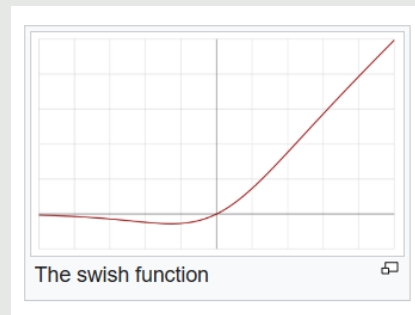
# Gated Linear Units

## GeGLU

$$\text{FFN}_{\text{GeGLU}}(x, W, V, W_2) = (\text{GELU}(xW) \otimes xV)W_2$$

## SwiGLU (swish is $x * \text{sigmoid}(x)$ )

$$\text{FFN}_{\text{SwiGLU}}(x, W, V, W_2) = (\text{Swish}_1(xW) \otimes xV)W_2$$



## Expts: GLUs

- Shazeer 2020

	Score Average	CoLA MCC	SST-2 Acc
FFN <sub>ReLU</sub>	83.80	51.32	94.04
FFN <sub>GELU</sub>	83.86	53.48	94.04
FFN <sub>Swish</sub>	83.60	49.79	93.69
FFN <sub>GLU</sub>	84.20	49.16	94.27
FFN <sub>GEGLU</sub>	84.12	53.65	93.92
FFN <sub>Bilinear</sub>	83.79	51.02	<b>94.38</b>
FFN <sub>SwiGLU</sub>	84.36	51.59	93.92
FFN <sub>ReGLU</sub>	<b>84.67</b>	<b>56.16</b>	<b>94.38</b>
[Raffel et al., 2019]	83.28	53.84	92.68
ibid. stddev.	0.235	1.111	0.569

# Expts

- Narang et al 2020

Model	Params	Ops	Step/s	Early loss	Final loss	SGLUE	XSum	WebQ
Vanilla Transformer	223M	11.1T	3.50	2.182 ± 0.005	1.838	71.66	17.78	23.02
GeLU	223M	11.1T	3.58	2.179 ± 0.003	1.838	<b>75.79</b>	<b>17.86</b>	<b>25.13</b>
Swish	223M	11.1T	3.62	2.186 ± 0.003	1.847	<b>73.77</b>	17.74	<b>24.34</b>
ELU	223M	11.1T	3.56	2.270 ± 0.007	1.932	67.83	16.73	23.02
GLU	223M	11.1T	3.59	2.174 ± 0.003	<b>1.814</b>	<b>74.20</b>	<b>17.42</b>	24.34
GeGLU	223M	11.1T	3.55	2.130 ± 0.006	<b>1.792</b>	<b>75.96</b>	<b>18.27</b>	<b>24.87</b>
ReGLU	223M	11.1T	3.57	2.145 ± 0.004	<b>1.803</b>	<b>76.17</b>	<b>18.36</b>	<b>24.87</b>
SeLU	223M	11.1T	3.55	2.315 ± 0.004	1.948	68.76	16.76	22.75
SwiGLU	223M	11.1T	3.53	2.127 ± 0.003	<b>1.789</b>	<b>76.00</b>	<b>18.20</b>	<b>24.34</b>
LiGLU	223M	11.1T	3.59	2.149 ± 0.005	<b>1.798</b>	<b>75.34</b>	<b>17.97</b>	<b>24.34</b>
Sigmoid	223M	11.1T	3.63	2.291 ± 0.019	1.867	<b>74.31</b>	17.51	23.02
Softplus	223M	11.1T	3.47	2.207 ± 0.011	1.850	<b>72.45</b>	17.65	<b>24.34</b>

# Why GLU?

ReLU/GELU give only **additive nonlinearities**.

GLUs introduce **multiplicative interactions**:

$$(W_1x)_i \times g_i(x)$$

This allows the network to represent functions like:

$$x_i \times x_j$$

which require multiple layers with ReLU.

So a GLU layer can represent richer feature interactions **in one layer**.

ReLU kills gradients when negative.

GELU is smoother but still saturates.

GLU variants maintain gradient flow because:

$$y = a(x) \times b(x)$$

Gradient:

$$\frac{\partial y}{\partial x} = a'(x)b(x) + a(x)b'(x)$$

Even if one term is small, the other may still carry gradient.

This reduces dead neurons and helps training stability.

# Serial vs Parallel Models

**Parallel Layers** – We use a “parallel” formulation in each Transformer block (Wang & Komatsuzaki, 2021), rather than the standard “serialized” formulation. Specifically, the standard formulation can be written as:

$$y = x + \text{MLP}(\text{LayerNorm}(x + \text{Attention}(\text{LayerNorm}(x))))$$

Whereas the parallel formulation can be written as:

$$y = x + \text{MLP}(\text{LayerNorm}(x)) + \text{Attention}(\text{LayerNorm}(x))$$

The parallel formulation results in roughly 15% faster training speed at large scales, since the MLP and Attention input matrix multiplications can be fused. Ablation experiments showed a small quality degradation at 8B scale but no quality degradation at 62B scale, so we extrapolated that the effect of parallel layers should be quality neutral at the 540B scale.

If implemented right, LayerNorm can be shared, and matrix multiplies can be fused

# Summary

- Pre vs Post Norm
  - Pre is a clear winner
- Layer vs RMSNorm
  - RMSNorm has better compute
- Activations
  - \*GLUs, but small differences
- Serial vs Parallel
  - Parallel has a compute improvement

AI Name	# Year	Norm	Parallel Layer	Pre-norm	Position embedding	Activations
Original transformer	2017	LayerNorm	Serial	<input type="checkbox"/>	Sine	ReLU
GPT	2018	LayerNorm	Serial	<input type="checkbox"/>	Absolute	GeLU
T5 (11B)	2019	RMSNorm	Serial	<input checked="" type="checkbox"/>	Relative	ReLU
GPT2	2019	LayerNorm	Serial	<input checked="" type="checkbox"/>	Absolute	GeLU
T5 (XXL 11B) v1.1	2020	RMSNorm	Serial	<input checked="" type="checkbox"/>	Relative	GeGLU
mT5	2020	RMSNorm	Serial	<input checked="" type="checkbox"/>	Relative	GeGLU
GPT3 (175B)	2020	LayerNorm	Serial	<input checked="" type="checkbox"/>	Absolute	GeLU
GPTJ	2021	LayerNorm	Parallel	<input checked="" type="checkbox"/>	RoPE	GeLU
LaMDA	2021			<input checked="" type="checkbox"/>	Relative	GeGLU
Anthropic LM (not claude)	2021			<input checked="" type="checkbox"/>		
Gopher (280B)	2021	RMSNorm	Serial	<input checked="" type="checkbox"/>	Relative	ReLU
GPT-NeoX	2022	LayerNorm	Parallel	<input checked="" type="checkbox"/>	RoPE	GeLU
BLOOM (175B)	2022	LayerNorm	Serial	<input checked="" type="checkbox"/>	Alibi	GeLU
OPT (175B)	2022	LayerNorm	Serial	<input checked="" type="checkbox"/>	Absolute	ReLU
PaLM (540B)	2022	RMSNorm	Parallel	<input checked="" type="checkbox"/>	RoPE	SwiGLU
Chinchilla	2022	RMSNorm	Serial	<input checked="" type="checkbox"/>	Relative	ReLU
Mistral (7B)	2023	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
LLaMA2 (70B)	2023	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
LLaMA (65B)	2023	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
GPT4	2023			<input type="checkbox"/>		
BaiChuan 2	2023	RMSNorm	Serial	<input checked="" type="checkbox"/>	Alibi	SwiGLU
Olmio 2	2024	RMSNorm	Serial	<input type="checkbox"/>	RoPE	SwiGLU
Gemma 2 (27B)	2024	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	GeGLU
Nemotron-4 (340B)	2024	LayerNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SigReLU
Qwen 2 (72B) - same for 2.5	2024	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
Falcon 2 11B	2024	LayerNorm	Parallel	<input checked="" type="checkbox"/>	RoPE	GeLU
Phi3 (small) - same for phi4	2024	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	GeGLU
Llama 3 (70B)	2024	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
Reka Flash	2024	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
Command R+	2024	LayerNorm	Parallel	<input checked="" type="checkbox"/>	RoPE	SwiGLU
OLMo	2024	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
Qwen (14B)	2024	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
DeepSeek (67B)	2024	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
Yi (34B)	2024	RMSNorm	Serial	<input checked="" type="checkbox"/>	RoPE	SwiGLU
Mixtral of Experts	2024			<input type="checkbox"/>		
Command A	2025	LayerNorm	Parallel	<input checked="" type="checkbox"/>	Hybrid (RoPE + NoF)	SwiGLU
Gemma 3	2025	RMSNorm	Serial	<input type="checkbox"/>	RoPE	GeGLU

# Other Variations

- Normalizations
- Activations, FFN
- Hyperparameters that matter/don't matter
- Stability Tricks

# Hyperparameters (Typical)

Transformer hyperparameter questions you might have had

- How much bigger should the feedforward size be compared to hidden size?
- How many heads, and should num\_heads always divide hidden size?
- What should my vocab size be?

And other model setting questions

- Do people even regularize these huge LMs?
- How do people scale these models - very deep or very wide?

# Consensus Hyperparameter #1

Feedforward – model dimension ratio.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

There are two dimensions that are relevant – the feedforward dim ( $d_{ff}$ ) and model dim ( $d_{model}$ ). What should their relationship be?

$$\begin{aligned} d_{ff} &= 65,536 \\ d_{model} &= 1024 \end{aligned} \quad \mathbf{T5}$$

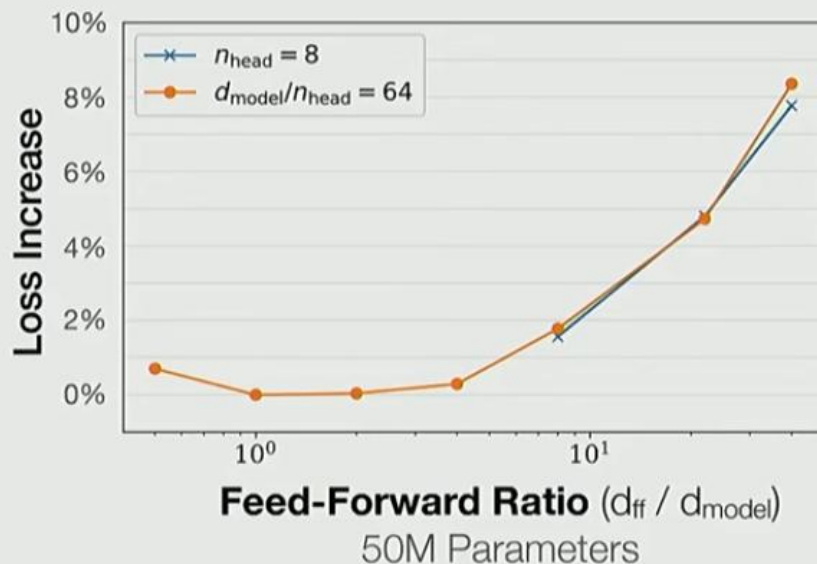
$$\mathbf{d_{ff} = 4 d_{model}}$$

Remember that GLU variants scale down by 2/3<sup>rd</sup>. This means most GLU variants have

$$d_{ff} = \frac{8}{3} d_{model}. \text{ This is mostly what happens.}$$

# Expts: [Kaplan 2020]

Empirically, there's a basin between 1-10 where this hyperparameter is near-optimal



That said, T5 has a follow-up model (T5 v1.1) that is 'improved' and uses a much more standard 2.5 multiplier on GeGLU, so the 64-times multiplier is likely suboptimal.

# Consensus Hyperparameter #2

## Multi-head self-attention is computationally efficient

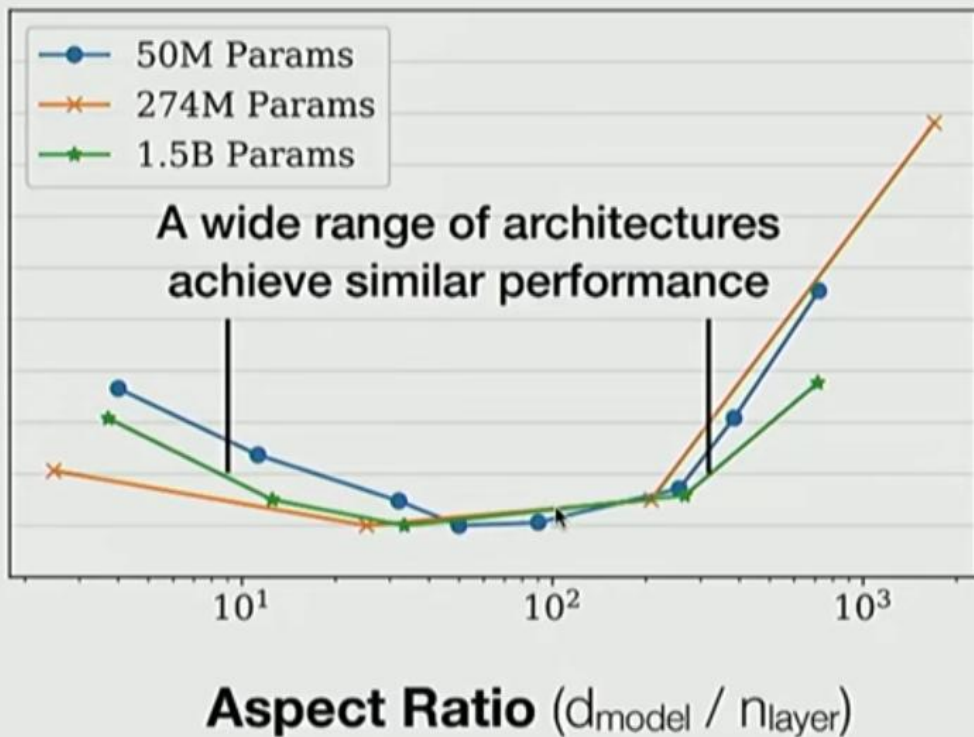
- Even though we compute  $h$  many attention heads, it's not really more costly.
  - We compute  $XQ \in \mathbb{R}^{n \times d}$ , and then reshape to  $\mathbb{R}^{n \times h \times d/h}$ . (Likewise for  $XK, XV$ .)
  - Then we transpose to  $\mathbb{R}^{h \times n \times d/h}$ ; now the head axis is like a batch axis.
  - Almost everything else is identical, and the **matrices are the same sizes**.

This doesn't *have to* be true: we can have head-dimensions  $>$  model-dim / num-head:

But most models do follow this guideline

# Aspect Ratio

- Should my model be deep or wide? Aspect ratio:  $d_{\text{model}}/n_{\text{layer}}$ 
  - sweet spot:  $\sim 128$



# Typical Vocabulary Sizes

- Monolingual models: 30k-50k
- Multilingual models: 100k-250k

# Dropout or Other Regularizations

## Arguments against:

- There is *a lot* of data (trillions of tokens), more than parameters.
- SGD only does a single pass on a corpus (hard to memorize)

This is all quite reasonable.. but what do people do in practice?

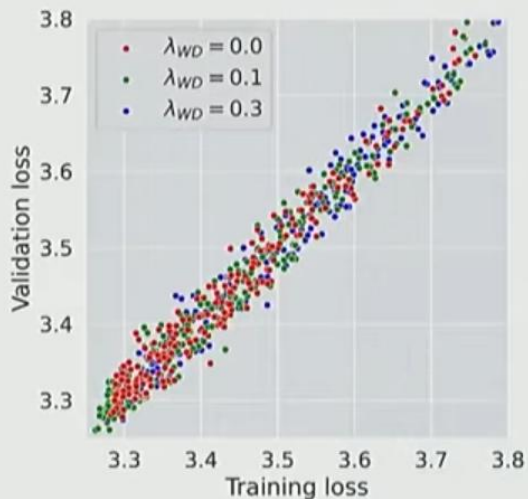
# In Practice

- Early days: lots of dropout
- These days: dropout out of fashion
- Weight Decay (of 0.1) continues to be important
  - Implemented in AdamW optimizer ( $\sim$ L2 regularization)

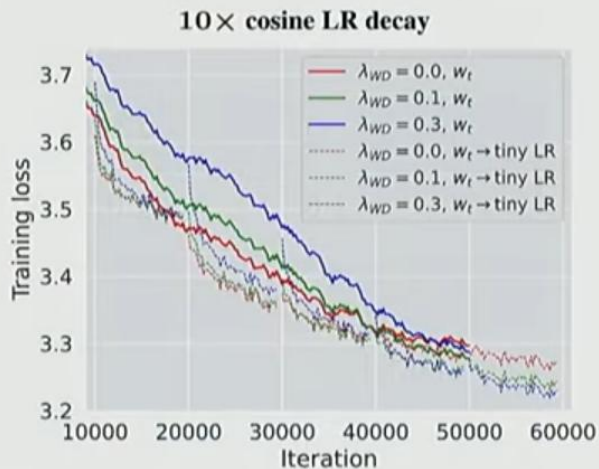
$$w_{t+1} = w_t - \eta (\hat{m}_t + \lambda w_t)$$

# Why Weight Decay LLMs?

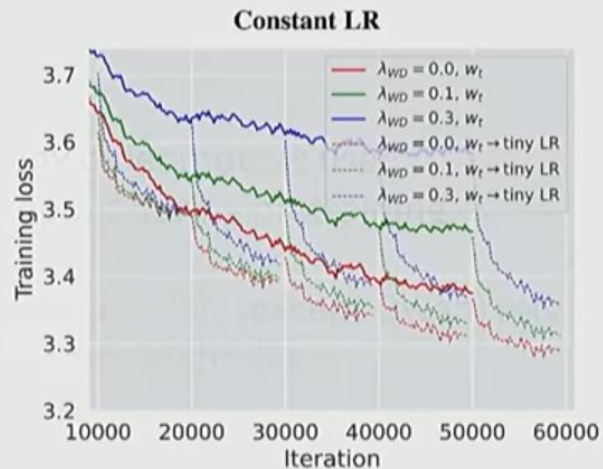
[Andriushchenko et al 2023] has interesting observations about LLM weight decay



It's not to control overfitting



Weight decay interacts with learning rates (cosine schedule)



# Summary

## Feedforward

- Factor-of-4 rule of thumb (8/3 for GLUs) is standard (with some evidence)

## Head dim

- Head dim\*Num head = D model is standard – but low to no validation

## Aspect ratio

- Wide range of ‘good’ values (100-200). Systems concerns dictate the value

## Regularization

- You still ‘regularize’ LMs but its effects are primarily on optimization dynamics

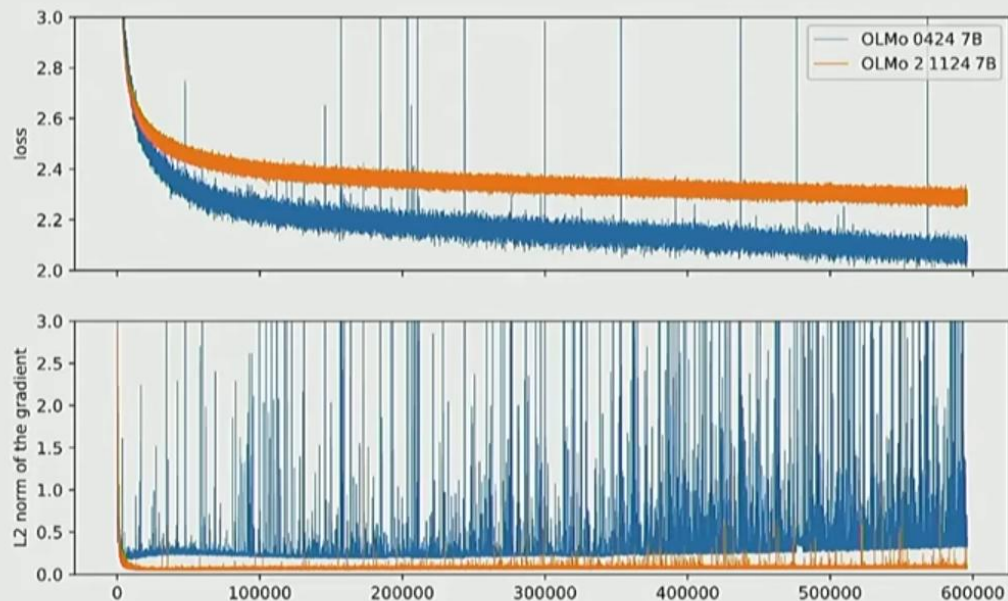
AI Name	Year	MLP factor	Aspect ratio (d/layer)	weight decay	drop_rate
Original transformer	2017	4	85	0	0.1
GPT	2018	4	64	0.1	0.1
T5 (11B)	2019	64	43	0	0.1
GPT2	2019	4	33	0.1	0.1
T5 (XXL 11B) v1.1	2020	2.5	171	0	0
mT5	2020	2.5	171	0	0
GPT3 (175B)	2020	4	128	0.1	0.1
GPT.J	2021		146	0.1	0
LaMDA	2021	8	128		
Anthropic LM (not claude)	2021	4	128		
Gopher (280B)	2021	4	205		
GPT-NeoX	2022	4	140	0.01	0
BLOOM (175B)	2022	4	205	0.1	0
OPT (175B)	2022	4	128	0.1	0.1
PaLM (540B)	2022	4	156		0
Chinchilla	2022	4	102		
Baichuan 2	2023	2.68	128	0.1	0
Mistral (7B)	2023	3.5	128	0.1	0
LLaMA2 (70B)	2023	3.5	102	0.1	0
LLaMA (65B)	2023	2.6875	102	0.1	0
GPT4	2023		0		
Ollmo 2	2024	2.6875	128		
Gemma 2 (27B)	2024	8	100		
Nemotron-4 (340B)	2024	4	192		0
Queen 2 (72b) - same for 2.5	2024	3.609	102		
Falcon 2 11B	2024	4	68	0.1	
Phi3 (small) - same for phi4	2024	3.5	128		
Llama 3 (70B)	2024	3.5	102		0
Reka Flash	2024		0		
Command R+	2024	2.75	192		
OLMo	2024	2.6875	128	0.1	0
Queen (14B)	2024	2.675	128	0.1	0.1
DeepSeek (67B)	2024	2.6875	86	0.1	0
Yi (34B)	2024	2.857142	119	0.1	0
Mistral of Experts	2024		0		
Command A	2025		0		

# Other Variations

- Normalizations
- Activations, FFN
- Hyperparameters that matter/don't matter
- Stability Tricks

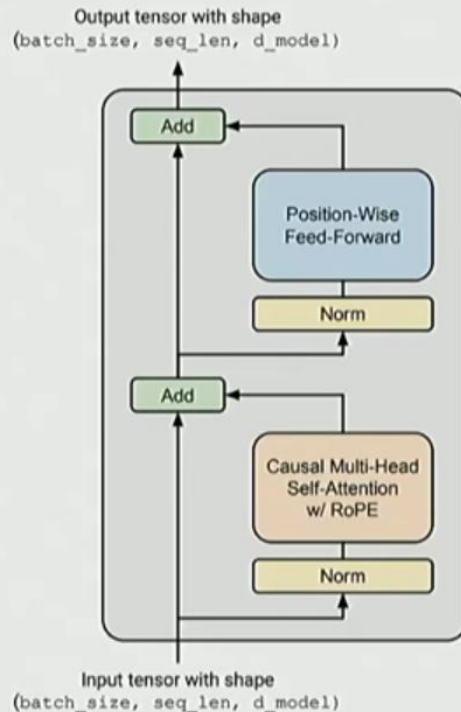
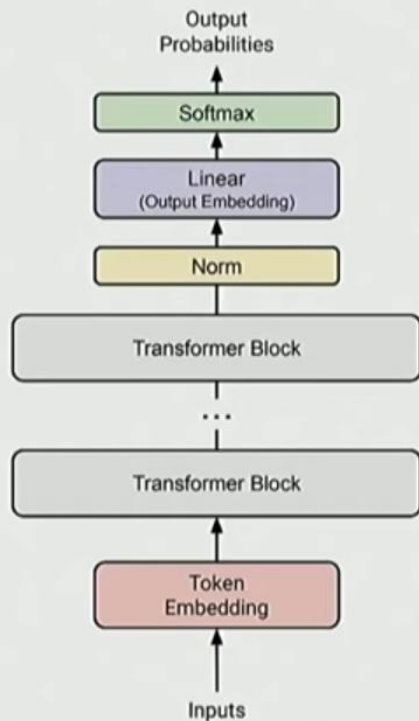
# Stability Tricks

Recently, lots of attention on *stable training*



Don't train models that look like the blue curve!

# Cause of Stability Issues



# Main Culprit

- **Softmax**
  - Exponential (numerically less well behaved)
  - Division by zero
- **Where are softmaxes**
  - Last layer (pre-generate)
  - MHA

# Z-Loss: Denominator Near 1

$$\begin{aligned}\log(P(x)) &= \log\left(\frac{e^{U_r(x)}}{Z(x)}\right) \\ &= U_r(x) - \log(Z(x)) \\ Z(x) &= \sum_{r'=1}^{|V|} e^{U_{r'}(x)}\end{aligned}$$

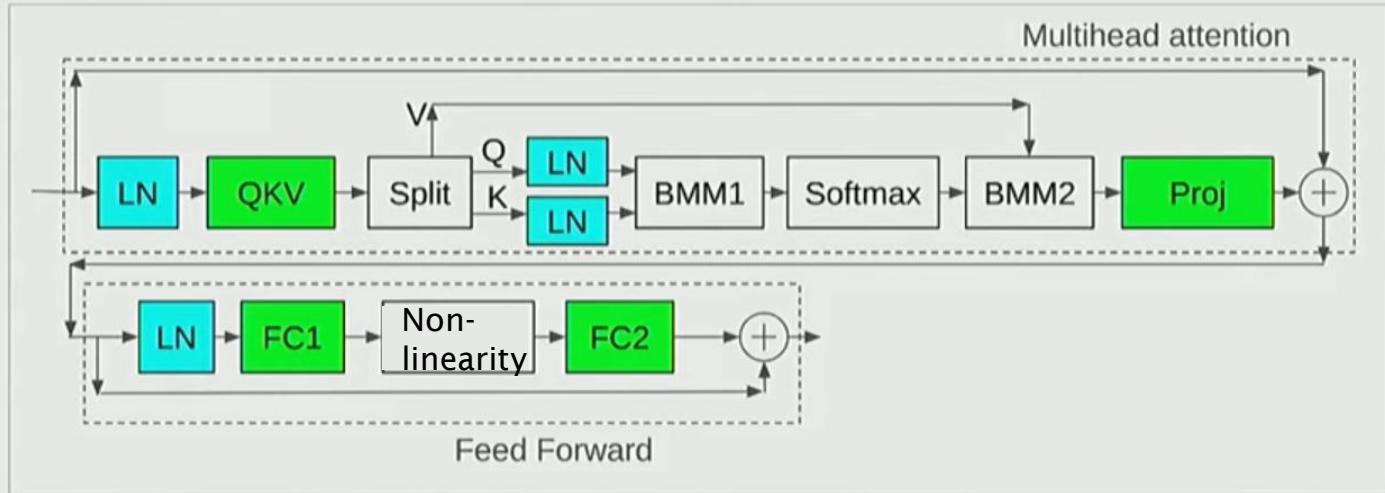
$$\begin{aligned}L &= \sum_i [\log(P(x_i)) - \alpha(\log(Z(x_i)) - 0)^2] \\ &= \sum_i [\log(P(x_i)) - \alpha \log^2(Z(x_i))]\end{aligned}$$

[From Devlin 2014]

This is useful for stability! PaLM pioneered this ‘z loss’ trick.

We additionally use an auxiliary loss of  $z\_loss = 10^{-4} \cdot \log^2 Z$  to encourage the softmax normalizer  $\log(Z)$  to be close to 0, which we found increases the stability of training.

# Attention Softmax Stability: QK Norm



The query and keys are Layer (RMS) normed before going into the softmax operation.