Lecture 3

Everything you didn't want to know about LM architecture and training

CS336

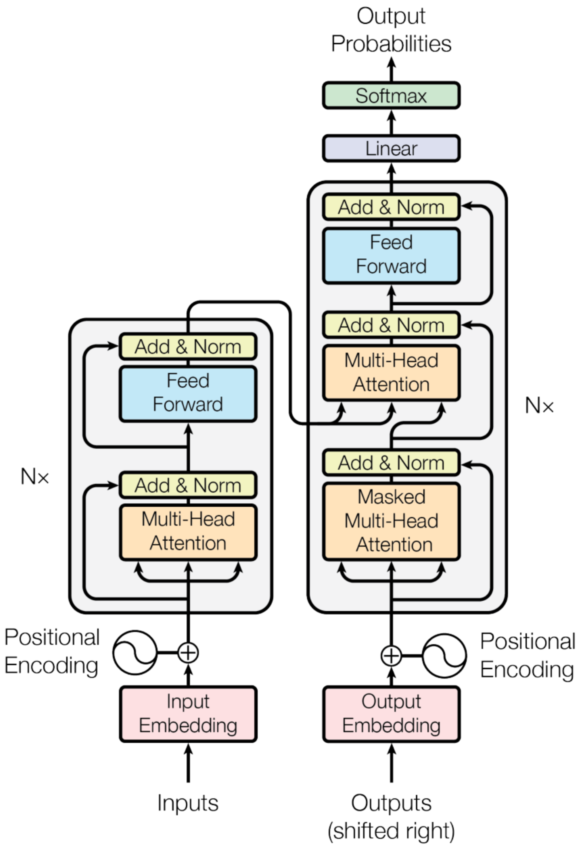
Tatsu H

Outline and goals

- Quick recap of the ‘standard’ transformer (what you implement)
- What do most of the large LMs have in common?
- What are common variations to the architecture / training process?

Today’s theme: the best way to learn is hands-on experience
the second best way is to try to learn from others’ experience

Starting point: the 'original' transformer



Review: choices in the standard transformer

Position embedding: sines and cosines

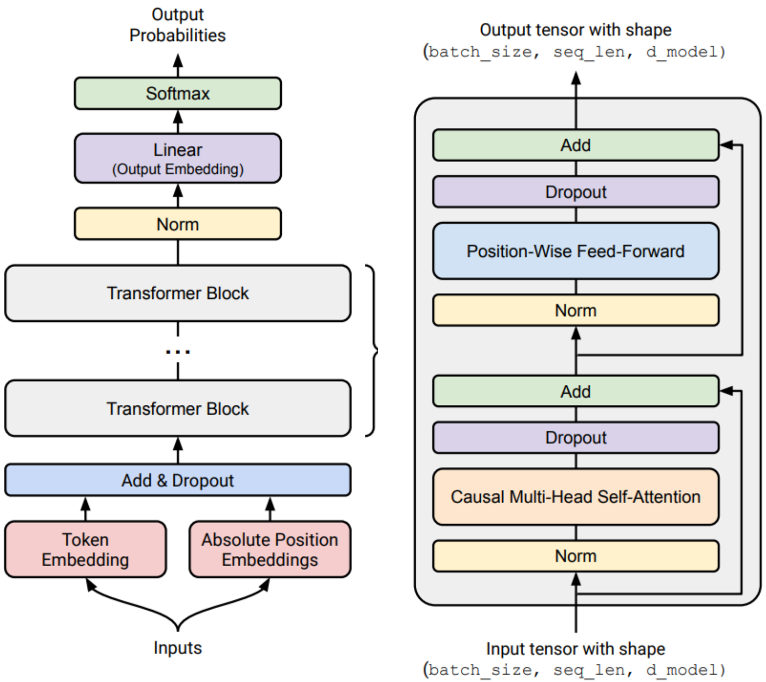
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

FFN: ReLU

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

Norm type: post-norm, LayerNorm

What you implemented – simple, modern variant



Differences:

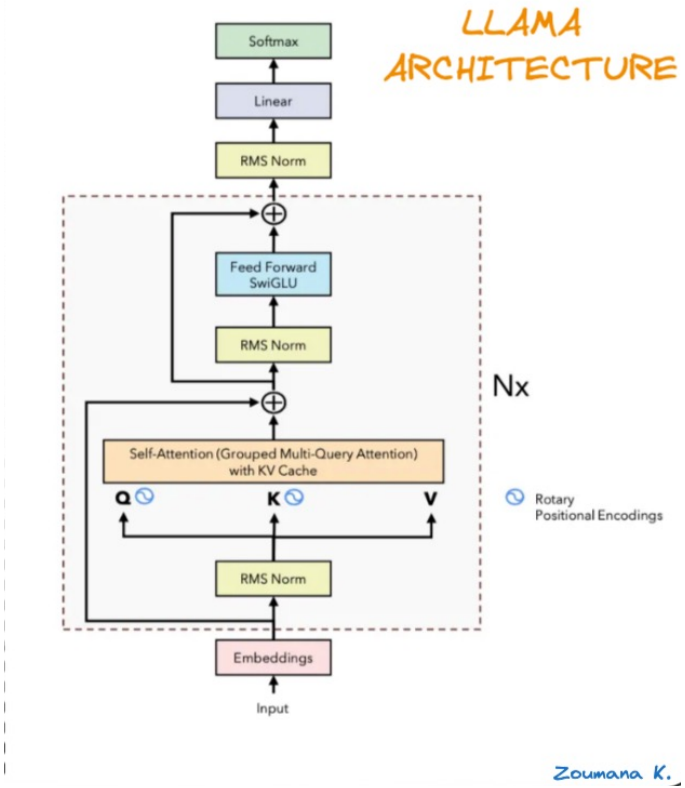
- **LayerNorm** is in front of the block
- **Absolute position embeddings**
- FF layers use **GeLU**, not ReLU
- Linear layers (and layernorm) have **no bias** (constant) terms

$$FFN(x) \coloneqq GELU(xW_1)W_2$$

Why did we pick these?

What should you pick?

What LLaMA (70B) does – yet more variations



Differences from your implementation:

- **Grouped Query Attention** (GQA)
- **Rotary embeddings** (??)
- **SwiGLU** (??)

What even are these things?

Fig from : https://www.youtube.com/watch?v=ISNdQcPhsts

How do we pick these?

Learn from the many other models (and papers) out there

Name	Has pa...	Link	Year	Tokenizer type	Vocab count	Norm	Parallel Layer	Pre-norm	Position embedding	Activations	MoE	MLP factor	num_layers	model_dim
Original transformer	Yes	arxiv.org/abs....03762	2017	BPE	37000	LayerNorm	Serial	☐	Sine	ReLU	☐	4	6	
GPT	Yes	cdn.openai.com/res...er.pdf	2018	BPE	40257	LayerNorm	Serial	☐	Absolute	GeLU	☐	4	12	
GPT2	Yes	cdn.openai.com/bet...rs.pdf	2019	BPE	50257	LayerNorm	Serial	☑	Sine	GeLU	☐	4	48	
T5 (11B)	Yes	arxiv.org/abs....10683	2019	SentencePiece	32128	RMSNorm	Serial	☑	Relative	ReLU	☐	64	24	
GPT3 (175B)	Yes	arxiv.org/abs....14165	2020	BPE	50257	LayerNorm	Serial	☑	Sine	GeLU	☐	4	96	
mT5	Yes	arxiv.org/abs....11934	2020	SentencePiece	250000	RMSNorm	Serial	☑	Relative	GeGLU	☐	2.5	24	
T5 (XXL 11B) v1.1	Kind of	github.com/goo...d#t511	2020	SentencePiece	32128	RMSNorm	Serial	☑	Relative	GeGLU	☐	2.5	24	
Gopher (280B)	Yes	arxiv.org/abs....11446	2021	SentencePiece	32000	RMSNorm	Serial	☑	Relative	ReLU	☐	4	80	
Anthropic LM (not claude)	Yes	arxiv.org/abs....00861	2021	BPE	65536			☑			☐	4	64	
LaMDA	Yes	arxiv.org/abs....08239	2021	BPE	32000			☑	Relative	GeGLU	☐	8	64	
GPTJ	Kind of	huggingface.co/Ele...t-j-6b	2021	BPE	50257	LayerNorm	Parallel	☑	RoPE	GeLU	☐		28	
Chinchilla	Yes	arxiv.org/abs....15556	2022	SentencePiece	32000	RMSNorm	Serial	☑	Relative	ReLU	☐	4	80	
PaLM (540B)	Yes	arxiv.org/abs....02311	2022	SentencePiece	256000	RMSNorm	Parallel	☑	RoPE	SwiGLU	☐	4	118	
OPT (175B)	Yes	arxiv.org/abs....01068	2022	BPE	50272	LayerNorm	Serial	☑	Absolute	ReLU	☐	4	96	
BLOOM (175B)	Yes	arxiv.org/abs....05100	2022	BPE	250680	LayerNorm	Serial	☑	AliBi	GeLU	☐	4	70	
GPT-NeoX	Yes	arxiv.org/pdf...45.pdf	2022	BPE	50257	LayerNorm	Parallel	☑	RoPE	GeLU	☐	4	44	
GPT4 OPEN	Ad	arxiv.org/abs....08774	2023	BPE	100000			☐			☐			
LLaMA (65B)	Yes	arxiv.org/abs....13971	2023	BPE	32000	RMSNorm	Serial	☑	RoPE	SwiGLU	☐	2.6875	80	
LLaMA2 (70B)	Yes	arxiv.org/abs....09288	2023	BPE	32000	RMSNorm	Serial	☑	RoPE	SwiGLU	☐	3.5	80	
Mistral (7B)	Yes	arxiv.org/abs....06825	2023	BPE	32000	RMSNorm	Serial	☑	RoPE	SwiGLU	☐	3.5	32	

We will talk through many major architecture and hyperparameter variants.

What do all these models have in common? What parts vary? What can we learn from this?

What are we going to cover?

Common architecture variations

- Activations, FFN
- Attention variants
- Position embeddings

Hyperparameters that (do or don’t) matter

- What is ff_dim? Do multi_head dims always sum to model_dim?

Tokenization variants (and pitfalls)

- How many tokens?

Optimization

Architecture variations..

Let's think about the core architecture piece

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

High level view:

- Low consensus (except pre-norm)
- Trends toward 'LLaMA-like' architectures

Pre-vs-post norm

The one thing *everyone* agrees on (in 2024)

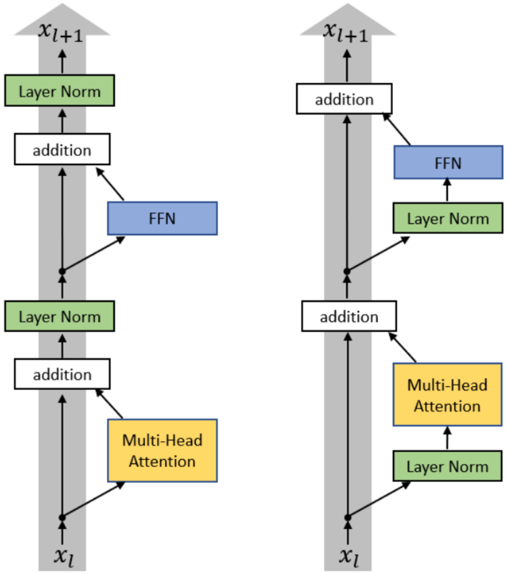


Figure from Xiong 2020

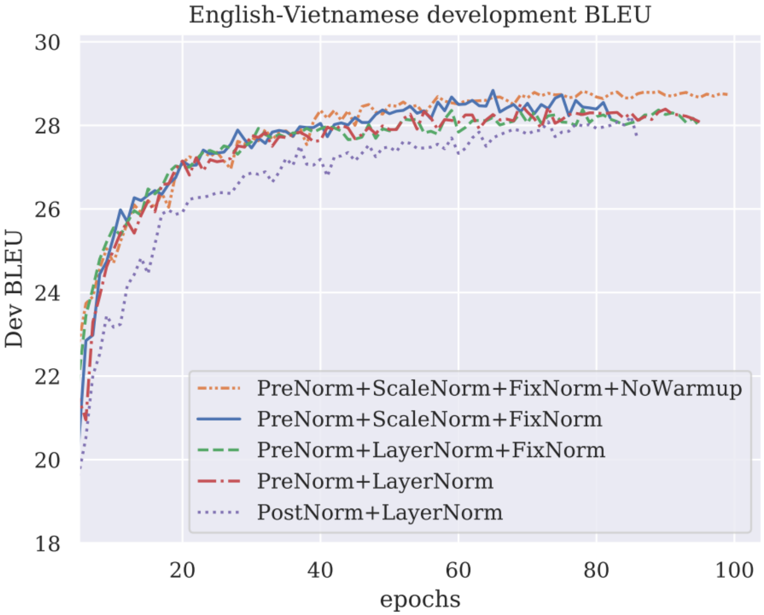
Post-LN Transformer	Pre-LN Transformer
$x^{post,1}_{l,i} = \text{MultiHeadAtt}(x^{post}_{l,i}, [x^{post}_{l,1}, \cdots, x^{post}_{l,n}])$	$x^{pre,1}_{l,i} = \text{LayerNorm}(x^{pre}_{l,i})$
$x^{post,2}_{l,i} = x^{post}_{l,i} + x^{post,1}_{l,i}$	$x^{pre,2}_{l,i} = \text{MultiHeadAtt}(x^{pre,1}_{l,i}, [x^{pre,1}_{l,1}, \cdots, x^{pre,1}_{l,n}])$
$x^{post,3}_{l,i} = \text{LayerNorm}(x^{post,2}_{l,i})$	$x^{pre,3}_{l,i} = x^{pre}_{l,i} + x^{pre,2}_{l,i}$
$x^{post,4}_{l,i} = \text{ReLU}(x^{post,3}_{l,i} W^{1,l} + b^{1,l}) W^{2,l} + b^{2,l}$	$x^{pre,4}_{l,i} = \text{LayerNorm}(x^{pre,3}_{l,i})$
$x^{post,5}_{l,i} = x^{post,3}_{l,i} + x^{post,4}_{l,i}$	$x^{pre,5}_{l,i} = \text{ReLU}(x^{pre,4}_{l,i} W^{1,l} + b^{1,l}) W^{2,l} + b^{2,l}$
$x^{post}_{l+1,i} = \text{LayerNorm}(x^{post,5}_{l,i})$	$x^{pre}_{l+1,i} = x^{pre,5}_{l,i} + x^{pre,3}_{l,i}$
	Final LayerNorm: $x^{pre}_{Final,i} \leftarrow \text{LayerNorm}(x^{pre}_{L+1,i})$

Set up LayerNorm so that it doesn't affect the main residual signal path (on the left)

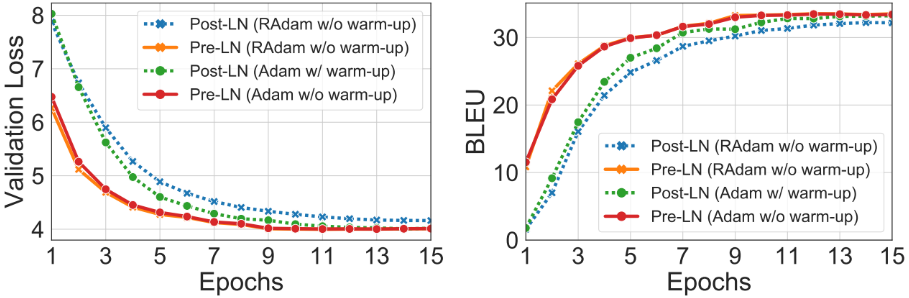
Almost all modern LMs use pre-norm (but BERT was post-norm)

(One somewhat funny exception – OPT350M. I don't know why this is post-norm)

Pre-vs-post-norm, the data

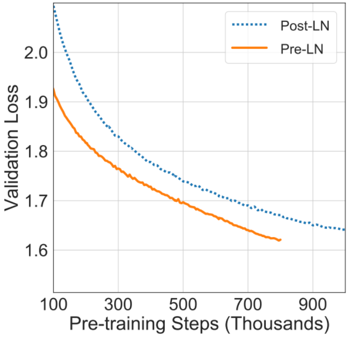


Salazar and Ngyuen 2019



(a) Validation Loss (IWSLT)

(b) BLEU (IWSLT)

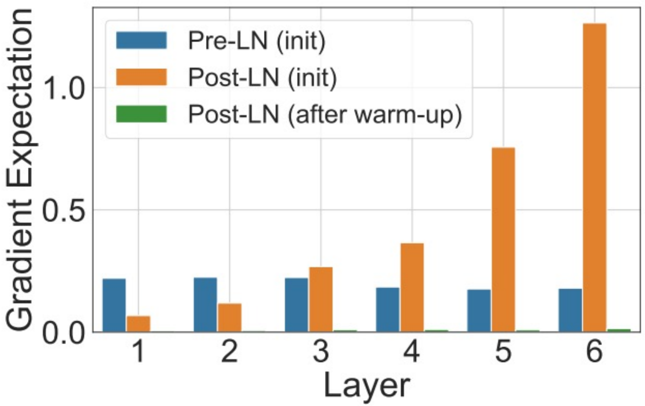


(a) Validation Loss on BERT

Figure from Xiong 2020

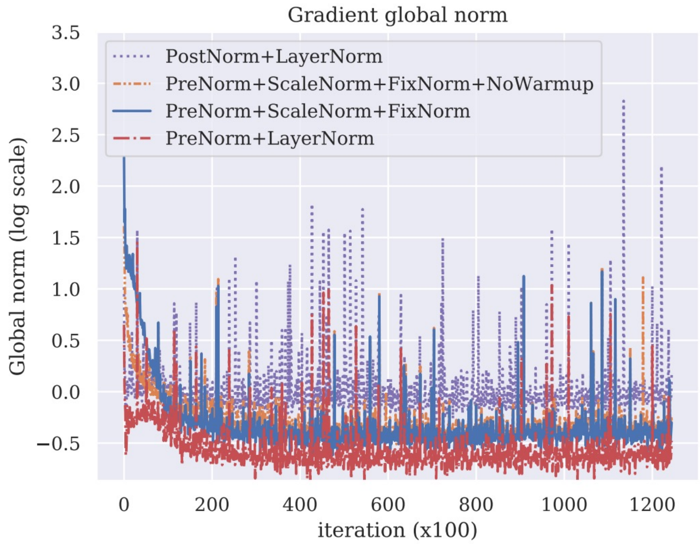
Pre-vs-post norm, explanations?

Gradient attenuation [Xiong 2020]



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

Gradient spikes [Salazar and Ngyuen]



Original stated advantage– removing warmup.

Today – stability and larger LRs for large networks

LayerNorm vs RMSNorm

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

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

Notable models:

GPT3/2/1, OPT, GPT-J, BLOOM

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

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

Notable models:

LLaMA-family, PaLM, Chinchilla, T5

Why RMSNorm?

Modern explanation – it's faster (and just as good).

- **Fewer operations** (no mean calculation)
- **Fewer parameters** (no bias term to store)

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

Does this explanation make sense?

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

Matrix multiplies are the *vast* majority of FLOPs (and memory)

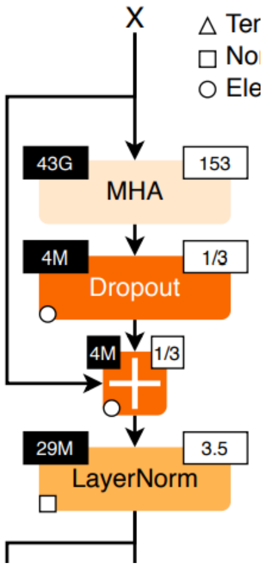
[Ivanov et al 2023]

Why RMSNorm (2)

Important lesson: FLOPS are not runtime! (we will discuss this in far more detail later)

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

RMSNorm can be faster because
it has fewer memory accesses



Left top (”43G”) is FLOPS
Right top (“153”) is the FLOP-to-memory ratio

[Ivanov et al 2023]

RMSNorm - validation

RMSNorm runtime (and surprisingly, perf) gains have been seen in papers

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	**1.821**	**75.45**	**17.94**	**24.07**	**27.14**
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

Narang et al 2020

More generally: dropping bias terms

Most modern transformers don't have bias terms.

Original Transformer:

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

Most implementations (if they're not gated):

$$FFN(x) = \sigma(xW_1)W_2$$

Reasons: memory (similar to RMSnorm) and optimization stability

LayerNorm: recap

- Basically everyone does pre-norm.
  - Intuition – keep the good parts of residual connections
  - Observations – nicer gradient propagation, fewer spike

- 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.

Activations

A whole zoo of activations ..

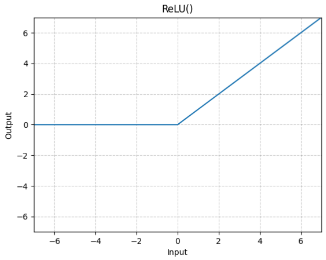
ReLU, GeLU, Swish, ELU, GLU, GeGLU, ReGLU, SeLU, SwiGLU, LiGLU

What are these things? What do people use? Does it matter?

A few of the common activations

ReLU

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

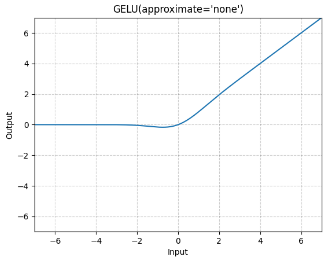


Notable models:

Original transformer, T5, Gopher, Chinchilla, OPT

GeLU

$$FF(x) = \mathrm{GELU}(xW_1)W_2$$
$$GELU(x) \coloneqq x\Phi(x)$$



Notable models:

GPT1/2/3, GPTJ, GPT-Neox, BLOOM

SwiGLU / GeGLU (next slide..)

Notable models:

Llama, PaLM, LaMDA, T5 v1.1, mT5

Gated activations (*GLU)

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

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

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

$$\mathrm{FF}_{\mathrm{ReGLU}}(x) = (\max(0, xW_1) \otimes xV)\, W_2$$

Gated variants of standard FF layers

GeGLU

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

Notable models:

T5 v1.1, mT5, LaMDA

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

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

Notable models:

LLaMa, PaLM

Note: Gated models use smaller dimensions for the d_{ff} by 2/3

Do gated linear units work?

Yes, fairly consistently so.

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

Shazeer 2020

Do gated linear units work (2)?

Yes, with other works corroborating Shazeer 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	**75.79**	**17.86**	**25.13**
Swish	$223M$	$11.1T$	3.62	2.186 ± 0.003	1.847	**73.77**	17.74	**24.34**
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	**1.814**	**74.20**	**17.42**	24.34
GeGLU	$223M$	$11.1T$	3.55	2.130 ± 0.006	**1.792**	**75.96**	**18.27**	**24.87**
ReGLU	$223M$	$11.1T$	3.57	2.145 ± 0.004	**1.803**	**76.17**	**18.36**	**24.87**
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	**1.789**	**76.00**	**18.20**	**24.34**
LiGLU	$223M$	$11.1T$	3.59	2.149 ± 0.005	**1.798**	**75.34**	**17.97**	**24.34**
Sigmoid	$223M$	$11.1T$	3.63	2.291 ± 0.019	1.867	**74.31**	17.51	23.02
Softplus	$223M$	$11.1T$	3.47	2.207 ± 0.011	1.850	**72.45**	17.65	**24.34**

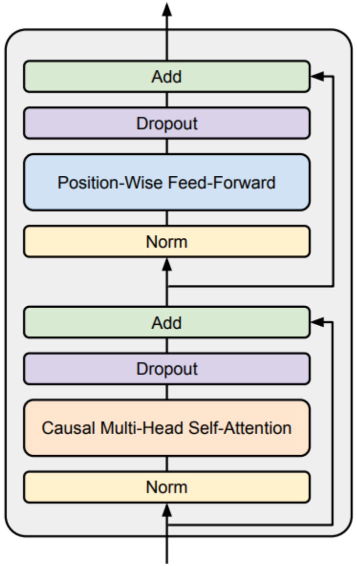
Narang et al 2020

Gating, activations

- Many variations (ReLU, GeLU, *GLU) across models.
- *GLU isn't necessary for a good model (see GPT3)
- But evidence points towards somewhat consistent gains from Swi/GeGLU

Serial vs Parallel layers

Normal transformer blocks are *serial* – they compute attention, then the MLP



Could we parallelize the transformer block?

Parallel layers

A few models (GPTJ, PaLM, GPT-NeoX) do parallel layers. Originally in GPT-J

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: architectures

Pre-vs-post norm:

- Everyone does pre-norm (except OPT350M), likely with good reason.

Layer vs RMSnorm:

- RMSnorm has clear compute wins, sometimes even performance

Gating:

- GLUs seem generally better, though differences are small

Serial vs parallel layers:

- No extremely serious ablations, but has a compute win.

Many variations in position embeddings

Sine embeddings: add sines and cosines that enable localization

$$Embed(x, i) = v_x + PE_{pos}$$

$$\boxed{\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{\text{model}}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{\text{model}}}) \end{aligned}}$$

Notable models:

Original transformer

Absolute embeddings: add a position vector to the embedding

$$Embed(x, i) = v_x + u_i$$

Notable models:

GPT1/2/3, OPT

Relative embeddings: add a vector to the *attention computation*

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}$$

Notable models:

T5, Gopher, Chinchilla

Rope embeddings (next slides..)

Notable models: GPTJ, PaLM, LLaMA

RoPE: rotary position embeddings

High level thought process: a *relative* position embedding should be some $f(x, i)$ s.t.

$$\langle f(x,i), f(y,j)\rangle = g(x, y, i-j)$$

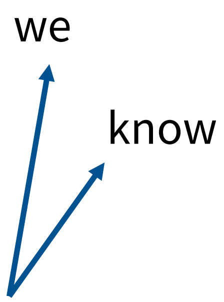
That is, the attention function *only* gets to depend on the relative position (i-j). How do existing embeddings not fulfill this goal?

- **Sine:** Has various cross-terms that are not relative
$$\langle Embed(x,i), Embed(y,i)\rangle = \langle v_x, v_y\rangle + \langle PE_i, v_y\rangle \ldots$$
- **Absolute:** obviously not relative
- **Relative embeddings:** $e_{ij} = \dfrac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}$ is not an inner product

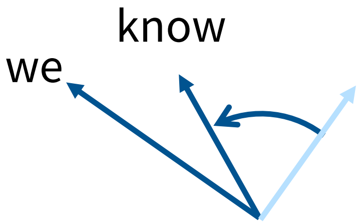
RoPE: rotary position embeddings

How can we solve this problem?

- We want our embeddings to be invariant to absolute position
- We know that inner products are invariant to arbitrary rotation.

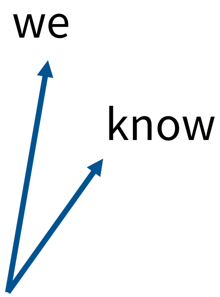


Position independent embedding



Embedding
“of course we know”

Rotate by ‘2 positions’

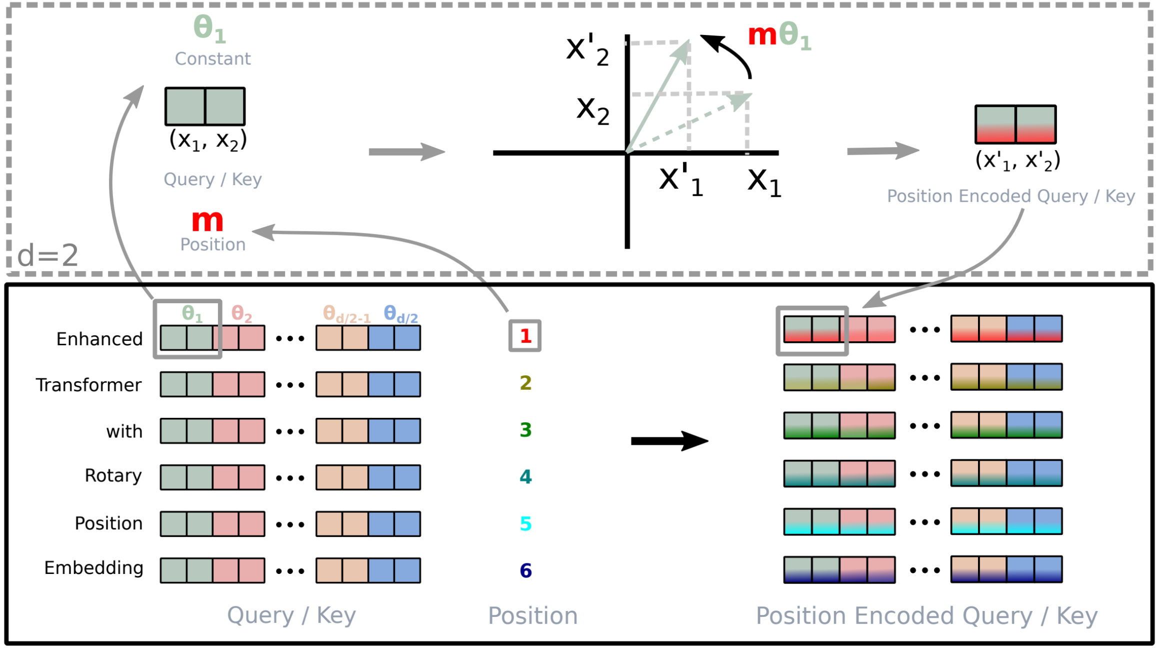


Embedding
“we know that”

Rotate by ‘0 positions’

RoPE: rotary position embeddings

There are many rotations, which one do you pick?



[Su et al 2021]

Just pair up the coordinates and rotate them in 2d (motivation: complex numbers)

The actual RoPE math

Multiply with sines and cosines

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m}\boldsymbol{W}_{\{q,k\}}\boldsymbol{x}_m \tag{14}$$

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix} \tag{15}$$

Difference with sine embeddings – not additive, no cross terms

Implementation and code for RoPE

Usual
attention stuff

```
query_states = self.q_proj(hidden_states)
key_states = self.k_proj(hidden_states)
value_states = self.v_proj(hidden_states)

# Flash attention requires the input to have the shape
# batch_size x seq_length x head_dim x hidden_dim
# therefore we just need to keep the original shape
query_states = query_states.view(bsz, q_len, self.num_heads, self.head_dim).transpose(1, 2)
key_states = key_states.view(bsz, q_len, self.num_key_value_heads, self.head_dim).transpose(1, 2)
value_states = value_states.view(bsz, q_len, self.num_key_value_heads, self.head_dim).transpose(1, 2)

cos, sin = self.rotary_emb(value_states, position_ids)
query_states, key_states = apply_rotary_pos_emb(query_states, key_states, cos, sin)
```

Get the RoPE
matrix cos/sin

Multiply
query/key inputs

…

Same stuff as the usual multi-head self attention below

Note: embedding at *each attention operation* to enforce position invariance

Hyperparameters

Transformer hyperparameter questions you might have had in 224n..

- How much bigger should the feedforward size be compared to hidden size?
- How many heads, and should num_heads always divide hidden size?

And other model setting questions

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

Surprising (?) consensus hyperparameter 1

Feedforward – model dimension ratio.

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

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

$$\boldsymbol{d_{ff} = 4\, d_{model}}$$

This is *almost always* true. There's just a few exceptions.

Exception #1 – GLU variants

Remember that GLU variants scale down by 2/3rd. This means most GLU variants have $d_{ff} = \frac{8}{3} d_{model}$. This is mostly what happens. Some notable such examples.

Model	d_{ff}/d_{model}
PaLM	4
Mistral 7B	3.5
LLaMA-2 70B	3.5
LLaMA 70B	2.68
Qwen 14B	2.67
DeepSeek 67B	2.68
Yi 34B	2.85
T5 v1.1	2.5

Models are roughly in this range, though PaLM, LLaMA2 and Mistral are slightly larger

Exception #2 – T5

As we have (and will) see, most LMs are have boring, conservative hyperparameters. One exception is T5 [Raffel et al 2020] which has some *very bold* settings.

In particular, for the 11B model, they set

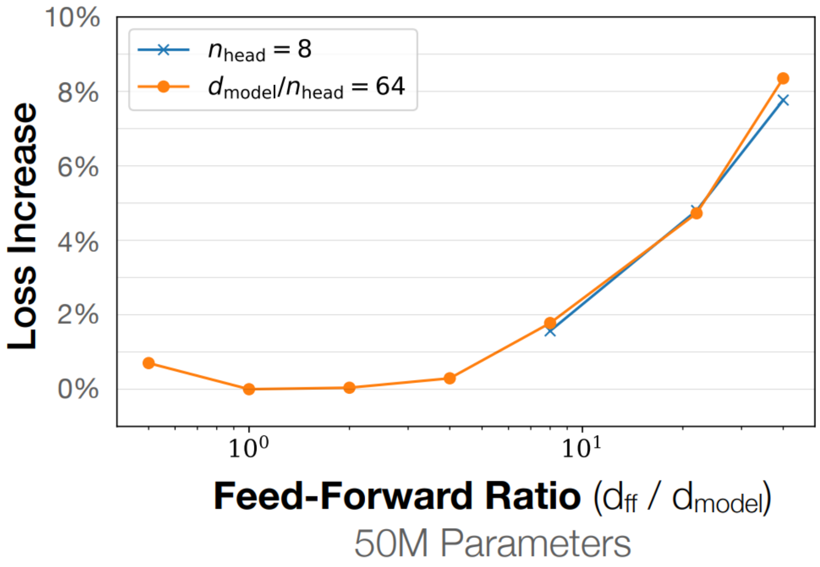
$$d_{ff} = 65{,}536$$
$$d_{model} = 1024$$

For an astounding 64-times multiplier.

> for "11B" we use $d_{\text{ff}} = 65{,}536$ with 128-headed attention producing a model with about 11 billion parameters. We chose to scale up d_{ff} specifically because modern accelerators (such as the TPUs we train our models on) are most efficient for large dense matrix multiplications like those in the Transformer's feed-forward networks.

Why this range of multipliers?

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



[Kaplan+ 2020]

What can we learn from the model-dim hyperparam?

- The ‘default’ choices of $d_{ff} = 4d_{model}$ and $d_{ff} = 2.66d_{model}$ have worked well for nearly all modern LLMs.

- But T5 does show that even radical choices of $d_{ff} = 64d_{model}$ can work. This hyperparameter choice isn’t written in stone.

- 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.

Surprising (?) consensus hyperparameter 2

Head-dim*num-heads to model-dim ratio. As a reminder, slide from 224n.

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-heads.

But most models do follow this guideline

How many heads, whats the model dim?

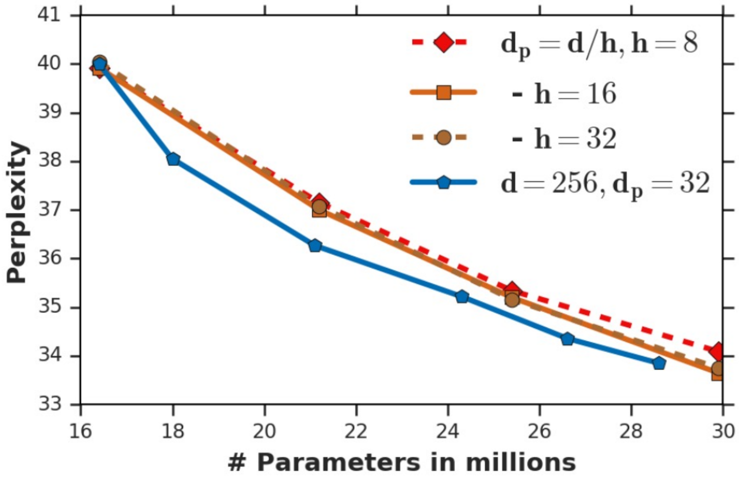
Some examples of this hyperparameter

	Num heads	Head dim	Model dim	Ratio
GPT3	96	128	12288	1
T5	128	128	1024	16
T5 v1.1	64	64	4096	1
LaMDA	128	128	8192	2
PaLM	48	258	18432	1.48
LLaMA2	64	128	8192	1

Most models have ratios around 1 – notable exceptions by some google models.

Evidence for 1-1 ratio?

There have been papers written against the 1-1 ratio [Bhojanapalli et al 2020]



But we don’t seem to be seeing significant ‘low rank bottlenecks’ in practice..

Aspect ratios

Should my model be deep or wide? *How* deep and how wide?

Most models are surprisingly consistent on this one too!

Sweet spot?

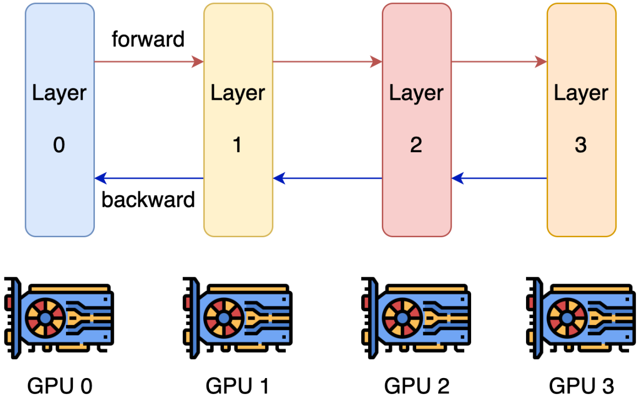
Model	d_{model}/n_{layer}
BLOOM	205
T5 v1.1	171
PaLM (540B)	156
GPT3/OPT/Mistral/Qwen	128
LLaMA / LLaMA2 / Chinchila	102
T5 (11B)	43
GPT2	33

Considerations about aspect ratio

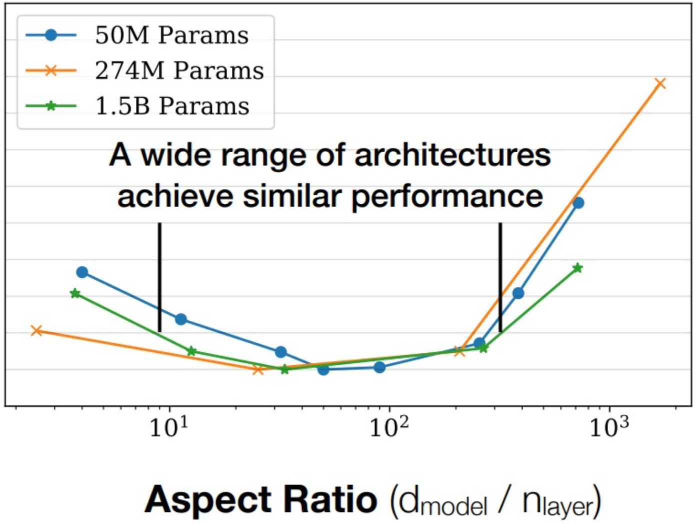
Extremely deep models are harder to parallelize

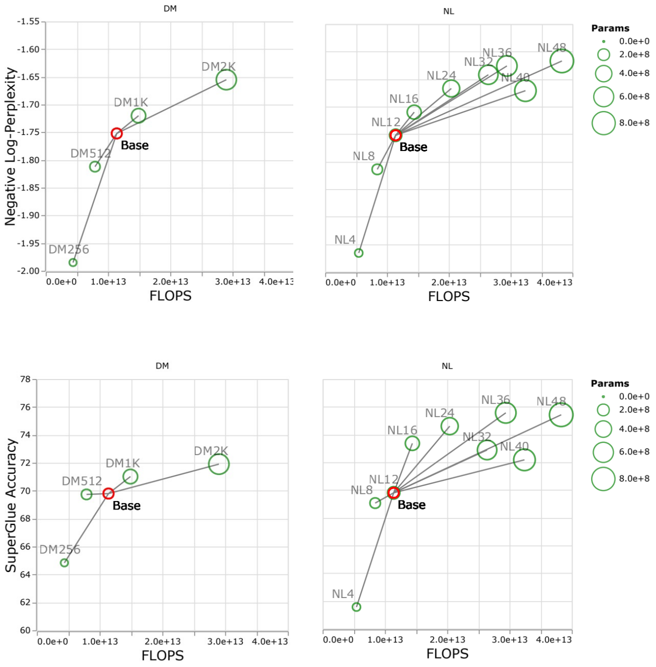
The Limits of Depth vs Width We note an obvious limitation with our advice. Scaling depth has an obvious limiter, i.e., they are non-parallelizable across different machines or devices and every computation has to always wait for the previous layer. This is unlike width, which can be easily parallelizable over thousands or hundreds of thousands of devices. Within the limitation of scaling

[Tay et al 2021]



Evidence on aspect ratio scaling





[Kaplan et al 2020]

[Tay et al 2021]

Dropout and other regularization

Do we need regularization during pretraining?

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?

Dropout and weight decay in practice

Model	Dropout*	Weight decay
Original transformer	0.1	0
GPT2	0.1	0.1
T5	0.1	0
GPT3	0.1	0.1
T5 v1.1	0	0
PaLM	0	(variable)
OPT	0.1	0.1
LLaMA	0	0.1
Qwen 14B	0.1	0.1

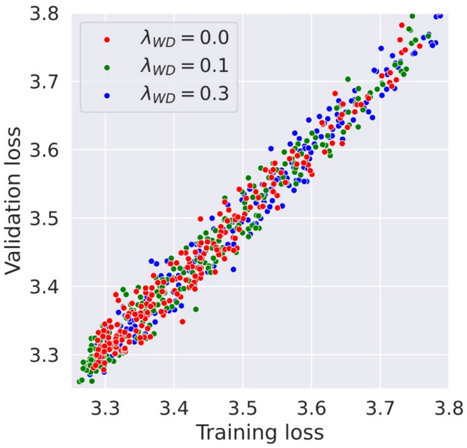
Many older models used dropout during pretraining

Newer models (except Qwen) rely only on weight decay

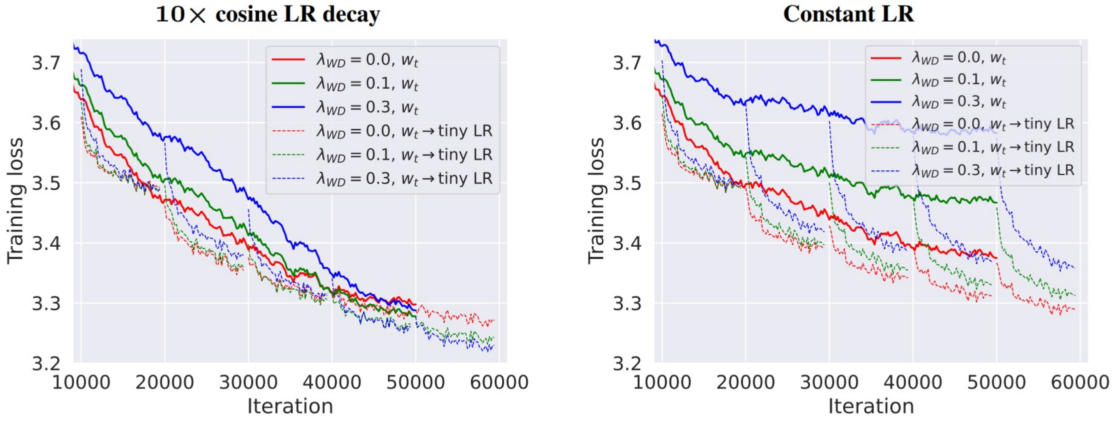
* Most of the times papers just don't discuss dropout. On open models, this closely matches not doing dropout. This may not be true of closed models.

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: hyperparameters

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

Tokenizers

The non-google world uses BPE. Google uses the SentencePiece library, which (sometimes) refers to a non-BPE subword tokenizer

Model	Tokenizer
Original transformer	BPE
GPT 1/2/3	BPE
T5 / mT5 / T5v1.1	SentencePiece (Unigram)
Gopher/Chinchilla	SentencePiece (??)
PaLM	SentencePiece (??)
LLaMA	BPE

Important property – all of these tokenizers are *invertible*

Sentencepiece

Open-source library with many subword tokenizers

Feature	SentencePiece	subword-nmt	WordPiece
Supported algorithm	BPE, unigram, char, word	BPE	BPE*
OSS?	Yes	Yes	Google internal
Subword regularization	Yes	No	No
Python Library (pip)	Yes	No	N/A
C++ Library	Yes	No	N/A
Pre-segmentation required?	No	Yes	Yes
Customizable normalization (e.g., NFKC)	Yes	No	N/A
Direct id generation	Yes	No	N/A

We will talk a bit about **normalization** and **unigram** subword tokenization

BPE and Unigram subword tokenizers

Algorithm 1 Byte-pair encoding (Sennrich et al., 2016; Gage, 1994)

Input: set of strings D, target vocab size k
procedure BPE(D, k)
    $V \leftarrow$ all unique characters in D
        (about 4,000 in English Wikipedia)
    while $|V| < k$ **do**   ▷ Merge tokens
        $t_L, t_R \leftarrow$ Most frequent bigram in D
        $t_{\text{NEW}} \leftarrow t_L + t_R$ ▷ Make new token
        $V \leftarrow V + [t_{\text{NEW}}]$
        Replace each occurrence of t_L, t_R in
            D with t_{NEW}
    end while
    return V
end procedure

Algorithm 2 Unigram LM (Kudo, 2018)

Input: set of strings D, target vocab size k
procedure UNIGRAMLM(D, k)
    $V \leftarrow$ all substrings occurring more than
        once in D (not crossing words)
    while $|V| > k$ **do**   ▷ Prune tokens
        Fit unigram LM θ to D
        for $t \in V$ **do**   ▷ Estimate token 'loss'
            $L_t \leftarrow p_\theta(D) - p_{\theta'}(D)$
            where θ' is the LM without token t
        end for
        Remove $\min(|V| - k, \lfloor \alpha |V| \rfloor)$ of the
        tokens t with highest L_t from V,
        where $\alpha \in [0, 1]$ is a hyperparameter
    end while
    Fit final unigram LM θ to D
    return V, θ
end procedure

BPE is 'bottom-up' (merge characters). Unigram is 'top-down' (prune substrings)

Unigram tokenizers

Original:	furiously	**Original:**	tricycles	**Original:**	nanotechnology
BPE:	_fur iously	**BPE:**	_t ric y cles	**BPE:**	_n an ote chn ology
Uni. LM:	_fur ious ly	**Uni. LM:**	_tri cycle s	**Uni. LM:**	_nano technology

Original:	Completely preposterous suggestions
BPE:	_Comple t ely _prep ost erous _suggest ions
Unigram LM:	_Complete ly _pre post er ous _suggestion s

Original:	corrupted	**Original:**	1848 and 1852,
BPE:	_cor rupted	**BPE:**	_184 8 _and _185 2,
Unigram LM:	_corrupt ed	**Unigram LM:**	_1848 _and _1852 ,

Original	磁性は様々に分類がなされている。							
BPE	磁	性は		様々	に分類		がなされている	。
Unigram LM	磁	性	は	様々	に	分類	がなされている	。
Gloss	magnetism		(top.)	various ways	in	classification	is done	.
Translation	Magnetism is classified in various ways.							

Some (Bostrom and Durrett 2020) have argued that BPE produces less semantic tokens.

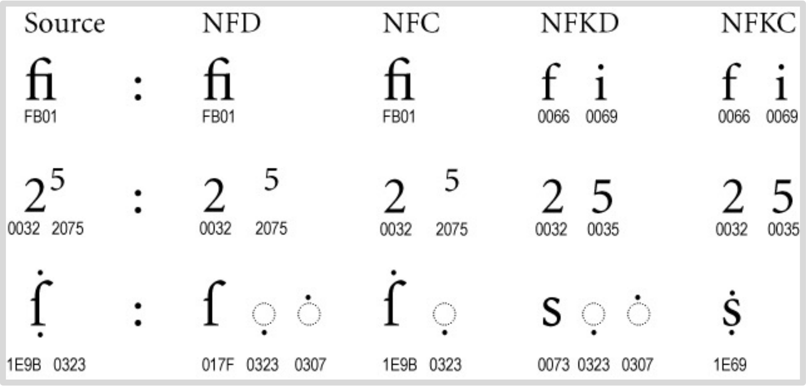
.. But BPE based LMs do work fine – the transformer on top can do quite a bit.

NFKC normalization

There are many characters that are different in Unicode but look very similar

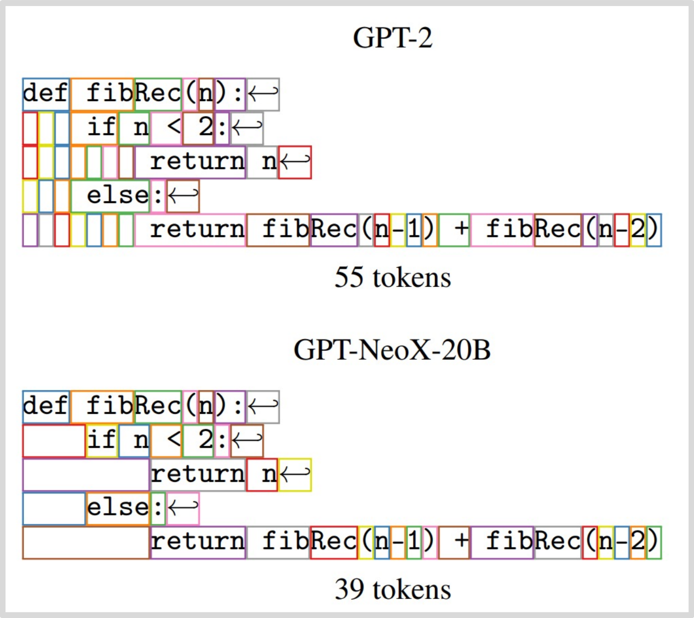
Roman ‘A’	Fullwidth ‘A’
A	Ａ

Some processing systems (e.g. sentencepiece) will NFKC normalize texts – with pros and cons



Whitespace and number related hacks

Multi-whitespace tokenization (GPT-NeoX)



Individual digit tokenization (LLaMA/DeepSeek)

> **Tokenizer.** We tokenize the data with the byte-pair encoding (BPE) algorithm (Sennrich et al., 2015), using the implementation from Sentence-Piece (Kudo and Richardson, 2018). Notably, we split all numbers into individual digits, and fallback to bytes to decompose unknown UTF-8 characters.

What are typical vocabulary sizes?

Monolingual models – 30-50k vocab

Model	Token count
Original transformer	37000
GPT	40257
GPT2/3	50257
T5/T5v1.1	32128
LLaMA	32000

Multilingual / production systems 100-250k

Model	Token count
mT5	250000
PaLM	256000
GPT4	100276
BLOOM	250680
DeepSeek	100000
Qwen 15B	152064
Yi	64000

Monolingual vocabs don't need to be huge, but multilingual ones do

Tokenizer: summary

- Everyone uses invertible subword tokenizers (BPE, Unigram) for good reason
- NFKC normalization is a double edged sword (2^5) and many models don't use it
- For math and code, careful manual handling of whitespace and numbers can help

Attention heads

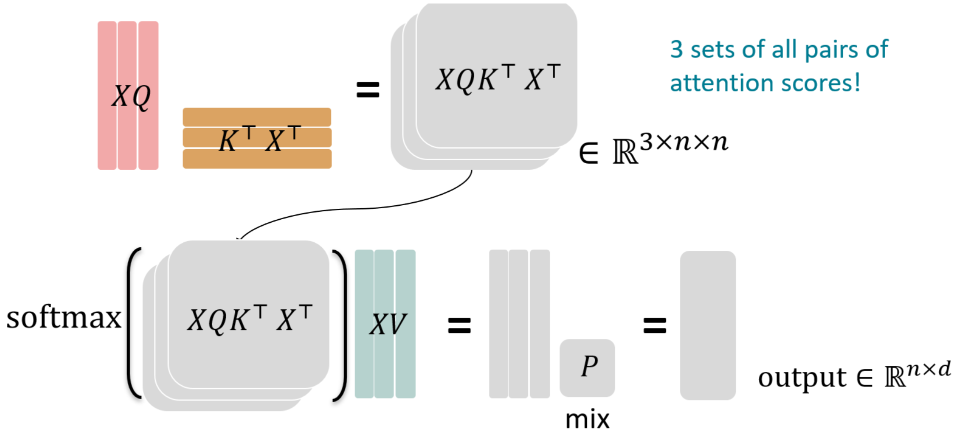
Most models don’t touch the attention head at all with a few minor exceptions..

GQA / MQA : Saving inference costs by reducing the number of heads

Sparse or sliding window attention (GPT4/Mistral): restricting the attention pattern to reduce compute cost

GQA/MQA – Reducing attention head cost

Let's think about the compute involved for attention



Total arithmetic operations (bnd^2), **total memory accesses** $(\underbrace{bnd}_{\text{X}} + \underbrace{bhn^2}_{\text{softmax}} + \underbrace{d^2}_{\text{projection}})$

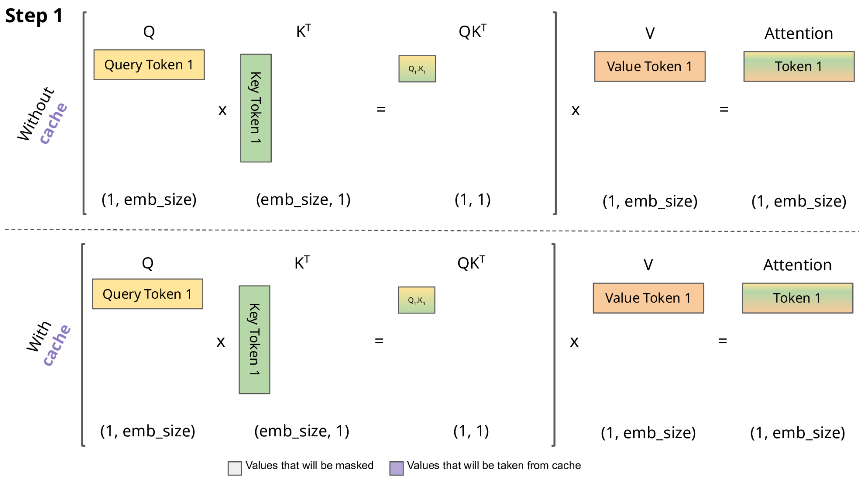
Arithmetic intensity is high $O\left(\left(\frac{1}{k}+\frac{1}{bn}\right)^{-1}\right)$ - we can keep our GPUs running

GQA/MQA – Reducing attention head cost

What about the *incremental* case when we generate text?

Key difference: can't parallelize the generation process – needs to be step by step

In this case – we need to incrementaly re-compute/update attention via the 'KV cache'



[Animation from https://medium.com/@joaolages/kv-caching-explained-276520203249]

GQA/MQA – Reducing attention head cost

What's the incremental arithmetic intensity?

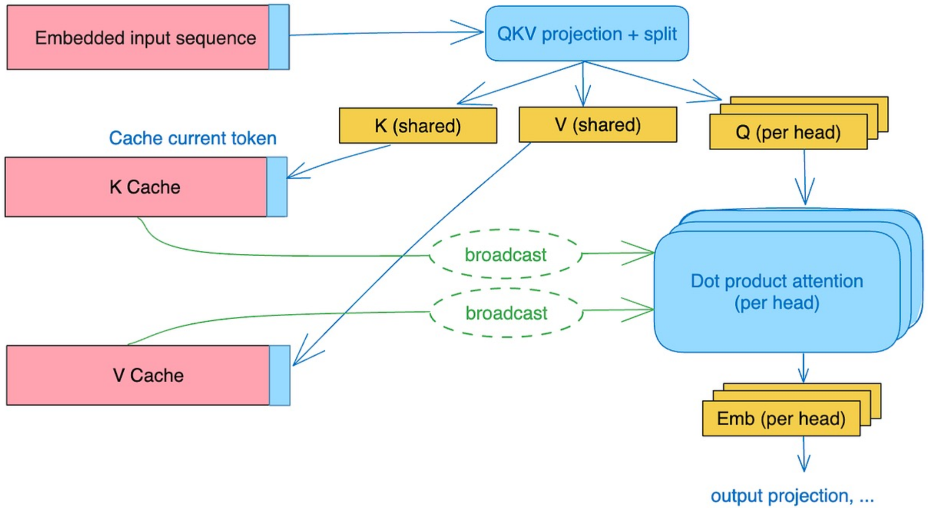
Total arithmetic operations (bnd^2), **total memory accesses** $(\overset{\text{X}}{bn^2d} + \overset{\text{projection}}{nd^2})$

Arithmetic intensity is not good $O\left(\left(\frac{n}{d}+\frac{1}{b}\right)^{-1}\right)$ - need large batches + short seq length (n) or big model dimensions (d)

Is there some way around this? The n/d term is difficult to reduce.

MQA – just have fewer key dimensions.

Key idea – have multiple queries, but just one dimension for keys and values



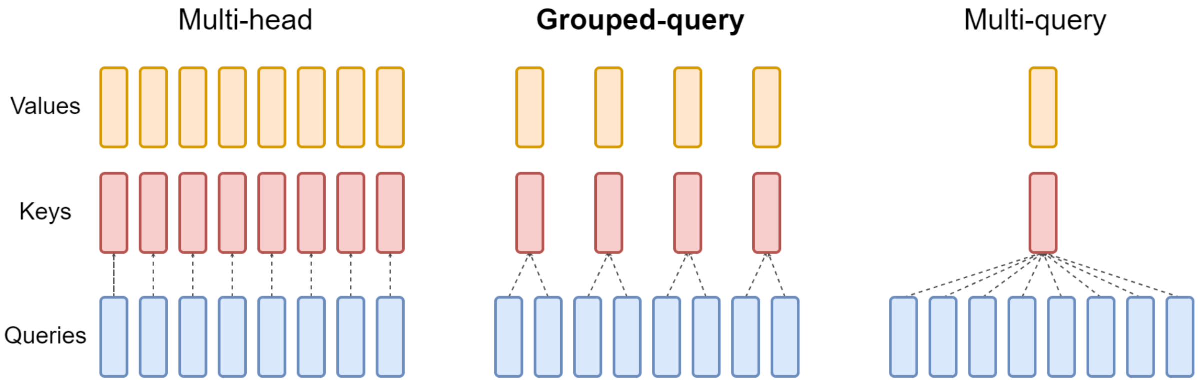
We have much fewer items to move in and out of memory (KV Cache)

Total memory access $(bnd + bn^2k + nd^2)$, **Arithmetic intensity** $O\left(\left(\frac{1}{d} + \frac{n}{dh} + \frac{1}{b}\right)^{-1}\right)$

[figure from https://blog.fireworks.ai/multi-query-attention-is-all-you-need-db072e758055]

Recent extension – GQA

Don't go all the way to one dimension of KV – have fewer dims



Simple knob to control expressiveness (key-query ratio) and inference efficiency

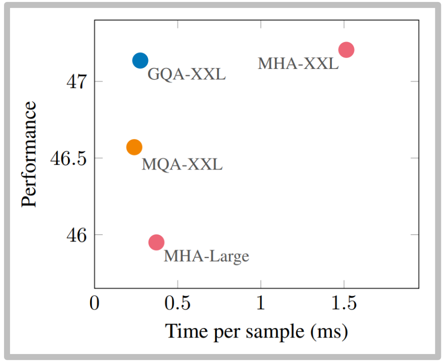
Does MQA hurt? Sometimes..

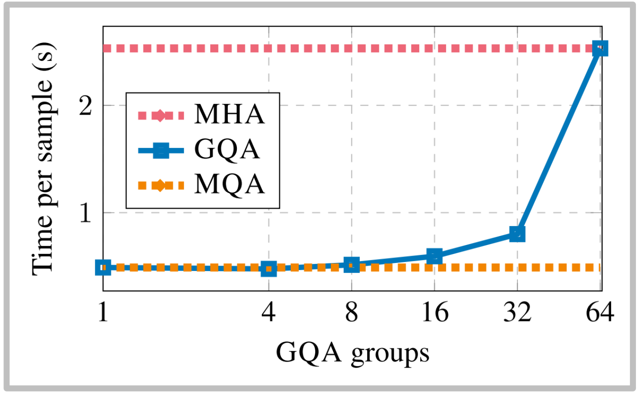
Small PPL hit w/ MQA [Shazeer 2019]

Table 3: Billion-Word LM Benchmark Results.

Attention	h	d_k, d_v	d_{ff}	dev-PPL
multi-head	8	128	8192	**29.9**
multi-query	8	128	9088	30.2
multi-head	1	128	9984	31.2
multi-head	2	64	9984	31.1
multi-head	4	32	9984	31.0
multi-head	8	16	9984	30.9

Low/no hit w/ GQA [Ainslie 2023]

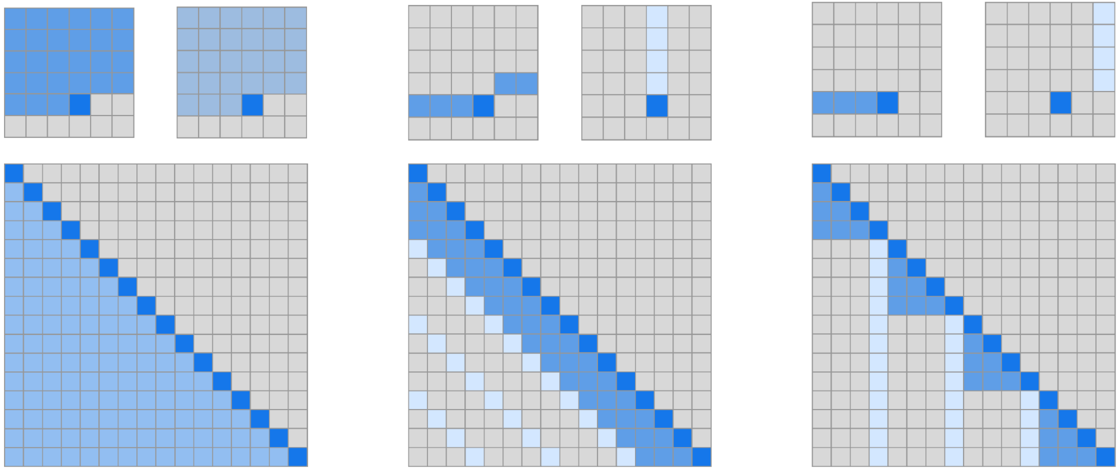




Sparse / sliding window attention

Attending to the entire context can be expensive (quadratic).

Build sparse / structured attention that trades off expressiveness vs runtime (GPT3)

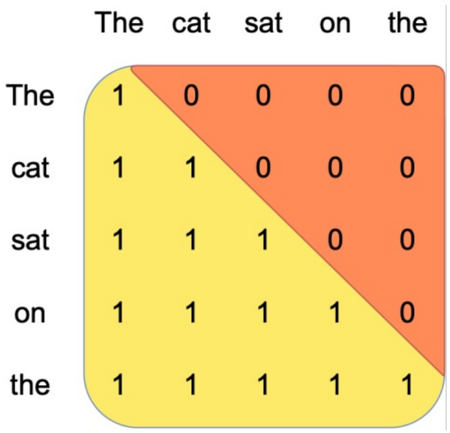


(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

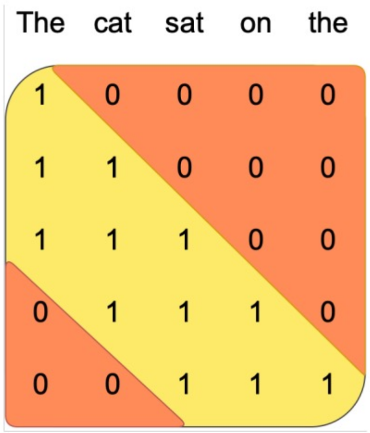
[Child et al 2019]

Sliding window attention

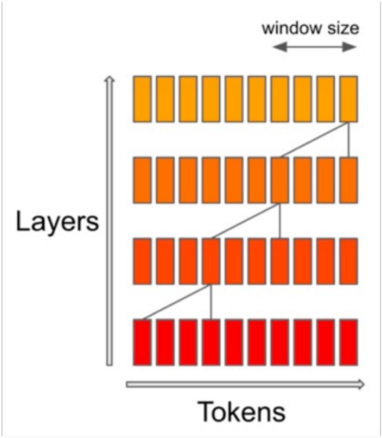
Another variation on this idea – sliding window attention



Vanilla Attention



Sliding Window Attention



Effective Context Length

Just use the main part of the strided pattern – let depth extend effective context (Mistral)

Recap, conclusion, etc.

Many aspects (arch, hparams) of transformers are in common across the big LMs

Name	Has pa...	Link	Year	Tokenizer type	Vocab count	Norm	Parallel Layer	Pre-norm	Position embedding	Activations	MoE	MLP factor	num_layers	model_dim
Original transformer	Yes	arxiv.org/abs....03762	2017	BPE	37000	LayerNorm	Serial	☐	Sine	ReLU	☐	4	6	
GPT	Yes	cdn.openai.com/res...er.pdf	2018	BPE	40257	LayerNorm	Serial	☐	Absolute	GeLU	☐	4	12	
GPT2	Yes	cdn.openai.com/bet...rs.pdf	2019	BPE	50257	LayerNorm	Serial	☑	Sine	GeLU	☐	4	48	
T5 (11B)	Yes	arxiv.org/abs....10683	2019	SentencePiece	32128	RMSNorm	Serial	☑	Relative	ReLU	☐	64	24	
GPT3 (175B)	Yes	arxiv.org/abs....14165	2020	BPE	50257	LayerNorm	Serial	☑	Sine	GeLU	☐	4	96	
mT5	Yes	arxiv.org/abs....11934	2020	SentencePiece	250000	RMSNorm	Serial	☑	Relative	GeGLU	☐	2.5	24	
T5 (XXL 11B) v1.1	Kind of	github.com/goo...d#t511	2020	SentencePiece	32128	RMSNorm	Serial	☑	Relative	GeGLU	☐	2.5	24	
Gopher (280B)	Yes	arxiv.org/abs....11446	2021	SentencePiece	32000	RMSNorm	Serial	☑	Relative	ReLU	☐	4	80	
Anthropic LM (not claude)	Yes	arxiv.org/abs....00861	2021	BPE	65536			☑			☐	4	64	
LaMDA	Yes	arxiv.org/abs....08239	2021	BPE	32000			☑	Relative	GeGLU	☐	8	64	
GPTJ	Kind of	huggingface.co/Ele...t-j-6b	2021	BPE	50257	LayerNorm	Parallel	☑	RoPE	GeLU	☐		28	
Chinchilla	Yes	arxiv.org/abs....15556	2022	SentencePiece	32000	RMSNorm	Serial	☑	Relative	ReLU	☐	4	80	
PaLM (540B)	Yes	arxiv.org/abs....02311	2022	SentencePiece	256000	RMSNorm	Parallel	☑	RoPE	SwiGLU	☐	4	118	
OPT (175B)	Yes	arxiv.org/abs....01068	2022	BPE	50272	LayerNorm	Serial	☑	Absolute	ReLU	☐	4	96	
BLOOM (175B)	Yes	arxiv.org/abs....05100	2022	BPE	250680	LayerNorm	Serial	☑	AliBi	GeLU	☐	4	70	
GPT-NeoX	Yes	arxiv.org/pdf...45.pdf	2022	BPE	50257	LayerNorm	Parallel	☑	RoPE	GeLU	☐	4	44	
GPT4	Ad	arxiv.org/abs....08774	2023	BPE	100000			☐			☐			
LLaMA (65B)	Yes	arxiv.org/abs....13971	2023	BPE	32000	RMSNorm	Serial	☑	RoPE	SwiGLU	☐	2.6875	80	
LLaMA2 (70B)	Yes	arxiv.org/abs....09288	2023	BPE	32000	RMSNorm	Serial	☑	RoPE	SwiGLU	☐	3.5	80	
Mistral (7B)	Yes	arxiv.org/abs....06825	2023	BPE	32000	RMSNorm	Serial	☑	RoPE	SwiGLU	☐	3.5	32	

Major differences? Position embeddings, activations, tokenization