Natural Language Processing with Deep Learning

CS224N/Ling284

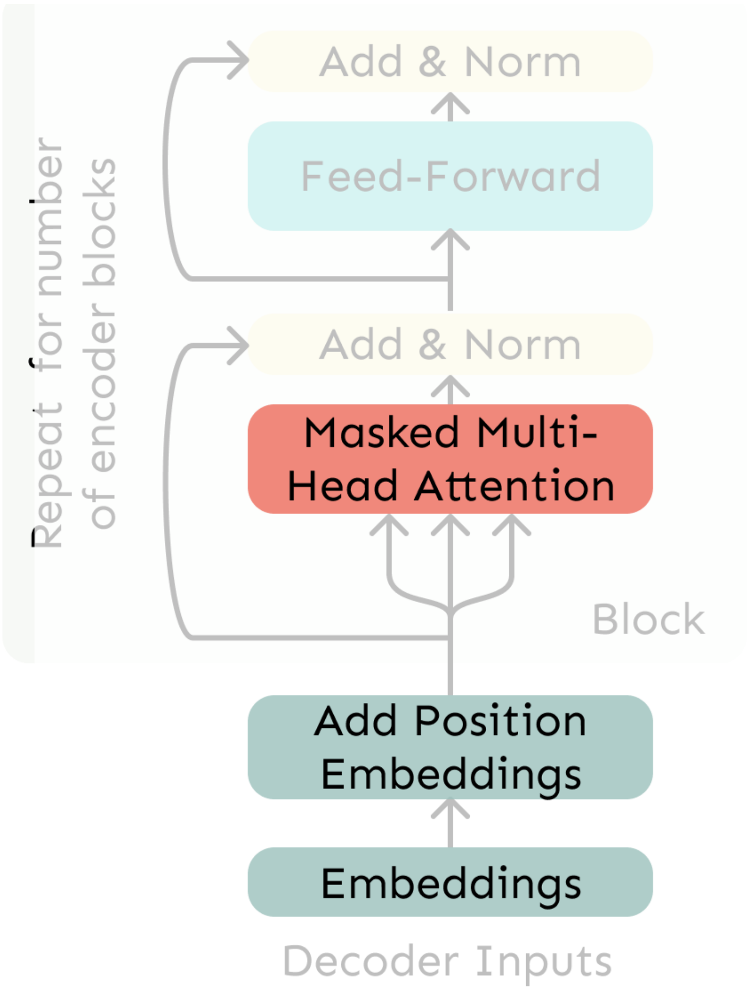
John Hewitt

Lecture 8: Self-Attention and Transformers

Adapted from slides by Anna Goldie, John Hewitt

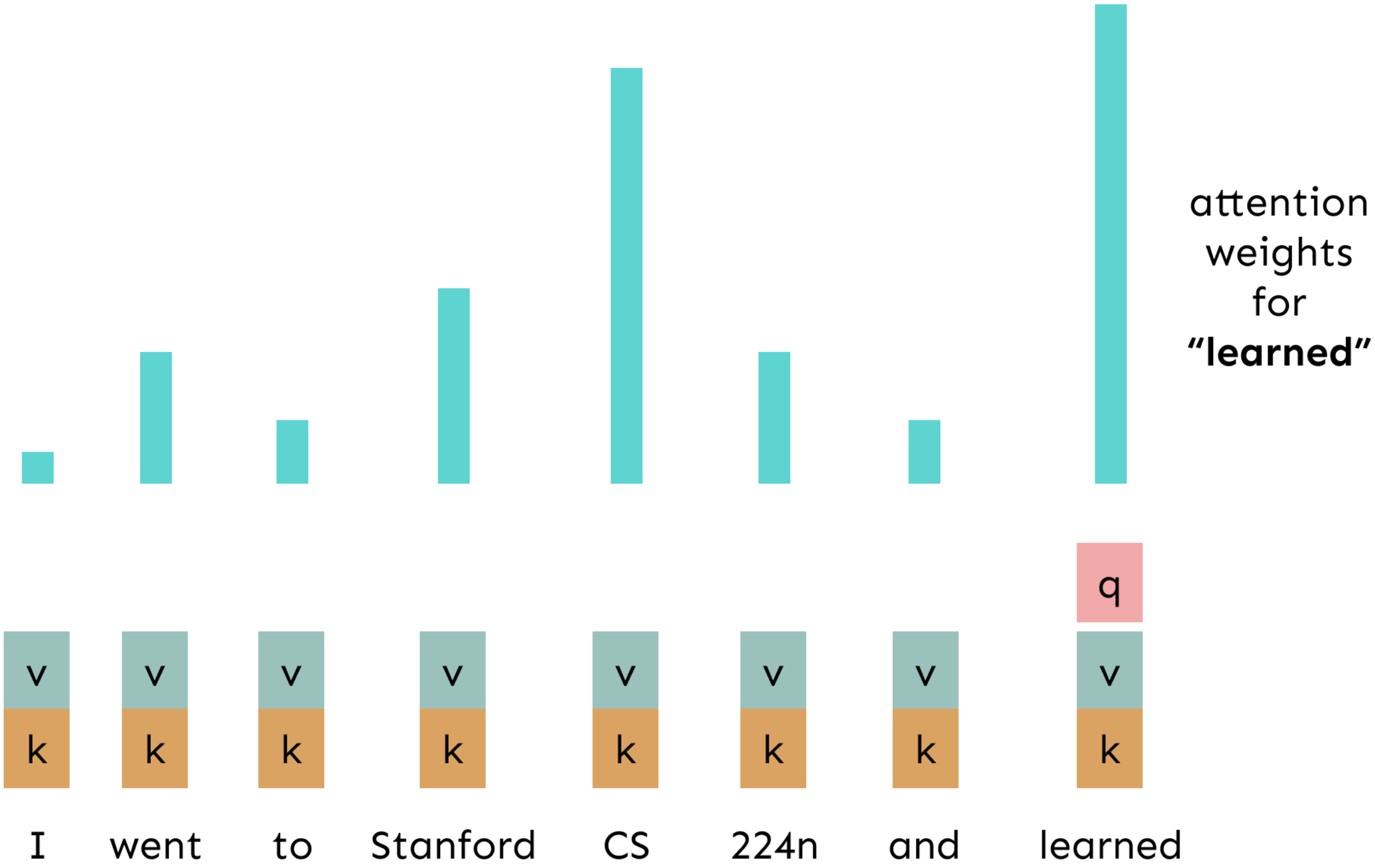
The Transformer Decoder

- A Transformer decoder is how we'll build systems like **language models**.
- It's a lot like our minimal self-attention architecture, but with a few more components.
- The embeddings and position embeddings are identical.
- We'll next replace our self-attention with **multi-head self-attention.**

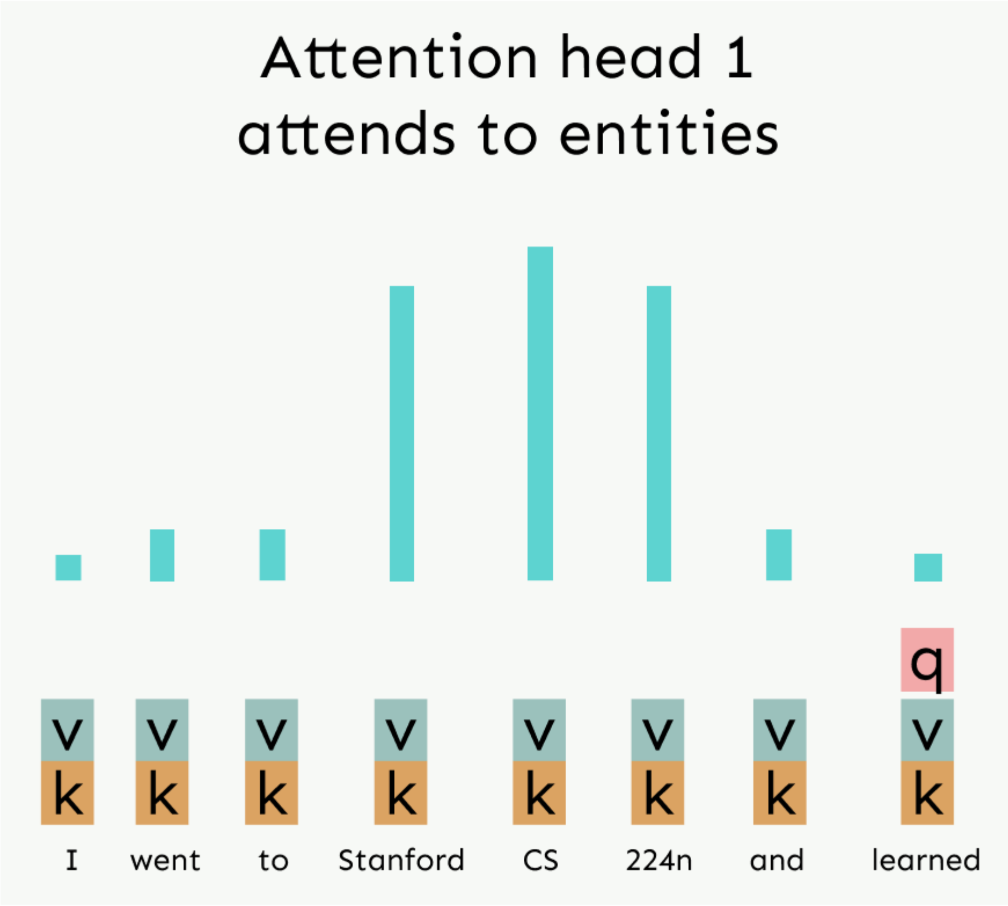


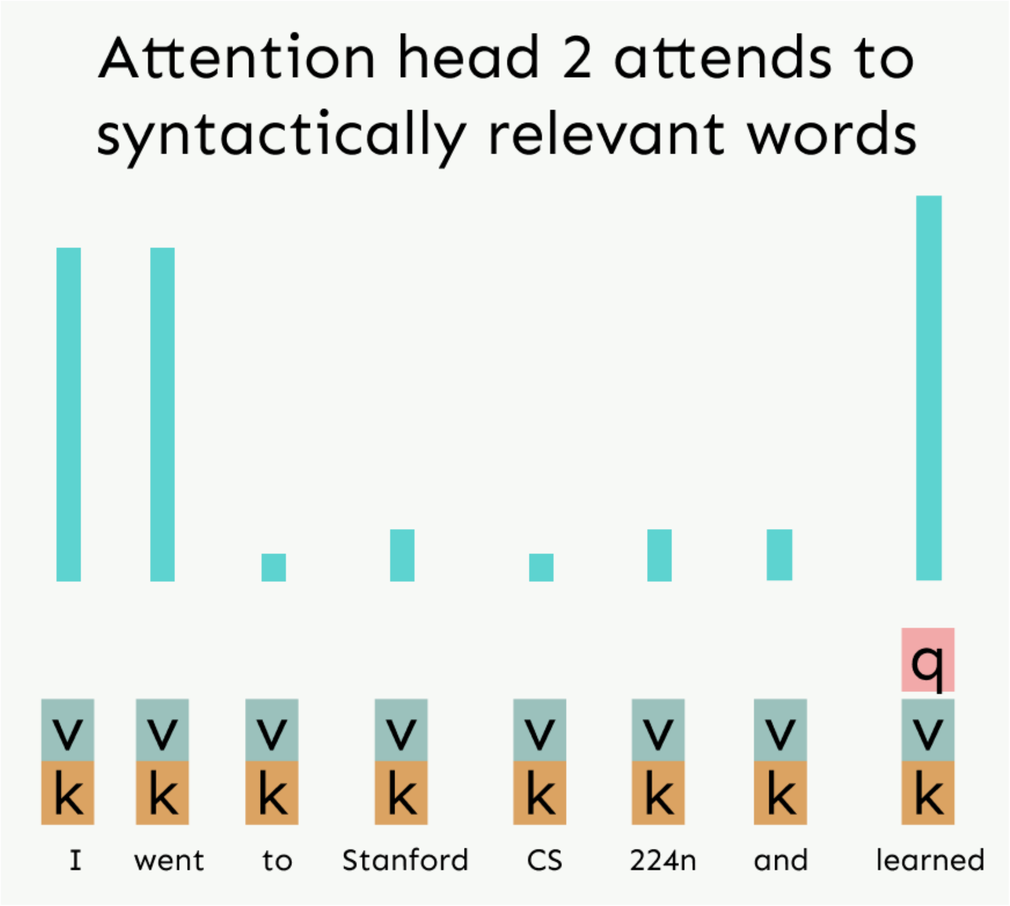
Transformer Decoder

Recall the Self-Attention Hypothetical Example



Hypothetical Example of Multi-Head Attention





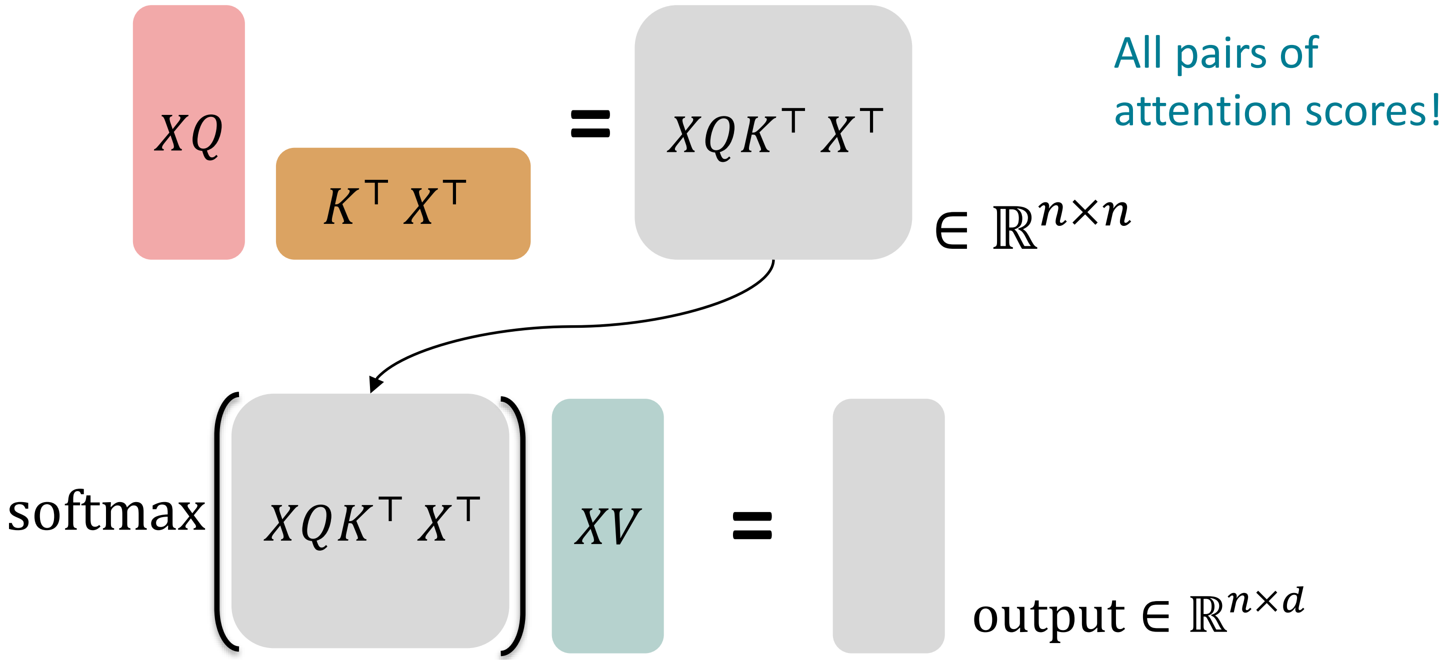
I went to Stanford CS 224n and learned

Sequence-Stacked form of Attention

- Let's look at how key-query-value attention is computed, in matrices.
  - Let $X = [x_1; \dots ; x_n] \in \mathbb{R}^{n \times d}$ be the concatenation of input vectors.
  - First, note that $XK \in \mathbb{R}^{n \times d}$, $XQ \in \mathbb{R}^{n \times d}$, $XV \in \mathbb{R}^{n \times d}$.
  - The output is defined as output $=$ $\mathrm{softmax}(XQ(XK)^\top)XV \in\in \mathbb{R}^{n \times d}$.

First, take the query-key dot products in one matrix multiplication: $XQ(XK)^\top$

Next, softmax, and compute the weighted average with another matrix multiplication.



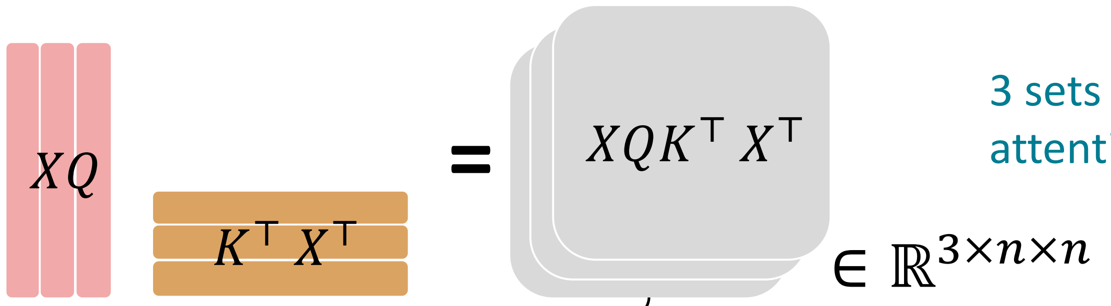
Multi-headed attention

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $x_i^\top Q^\top K x_j$ is high, but maybe we want to focus on different j for different reasons?
- We'll define **multiple attention "heads"** through multiple Q,K,V matrices
- Let, $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and ℓ ranges from 1 to h.
- Each attention head performs attention independently:
  - $\text{output}_\ell = \text{softmax}\left(X Q_\ell K_\ell^\top X^\top\right) * X V_\ell$, where $\text{output}_\ell \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
  - $\text{output} = [\text{output}_1; \ldots; \text{output}_h] Y$, where $Y \in \mathbb{R}^{d \times d}$
- Each head gets to "look" at different things, and construct value vectors differently.

Multi-head self-attention is computationally efficient

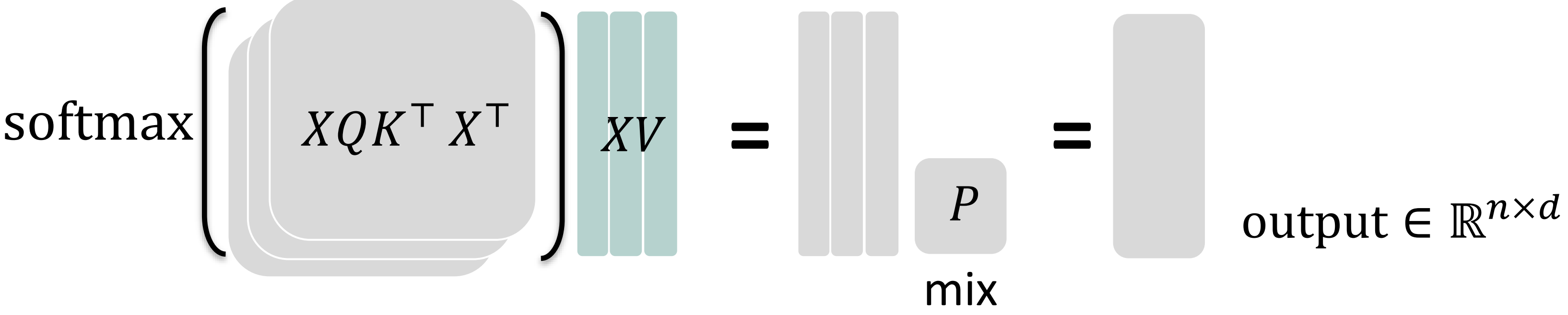
- Even though we compute h many attention heads, it's not really more costly.
  - We compute $XQ \in \mathbb{R}^{n \times d}$, and then reshape to $\mathbb{R}^{n \times h \times d/h}$. (Likewise for XK, XV.)
  - Then we transpose to $\mathbb{R}^{h \times n \times d/h}$; now the head axis is like a batch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

First, take the query-key dot products in one matrix multiplication: $XQ(XK)^\top$



3 sets of all pairs of attention scores!

Next, softmax, and compute the weighted average with another matrix multiplication.



Scaled Dot Product [Vaswani et al., 2017]

- **"Scaled Dot Product"** attention aids in training.
- When dimensionality d becomes large, dot products between vectors tend to become large.
  - Because of this, inputs to the softmax function can be large, making the gradients small.
- Instead of the self-attention function we've seen:

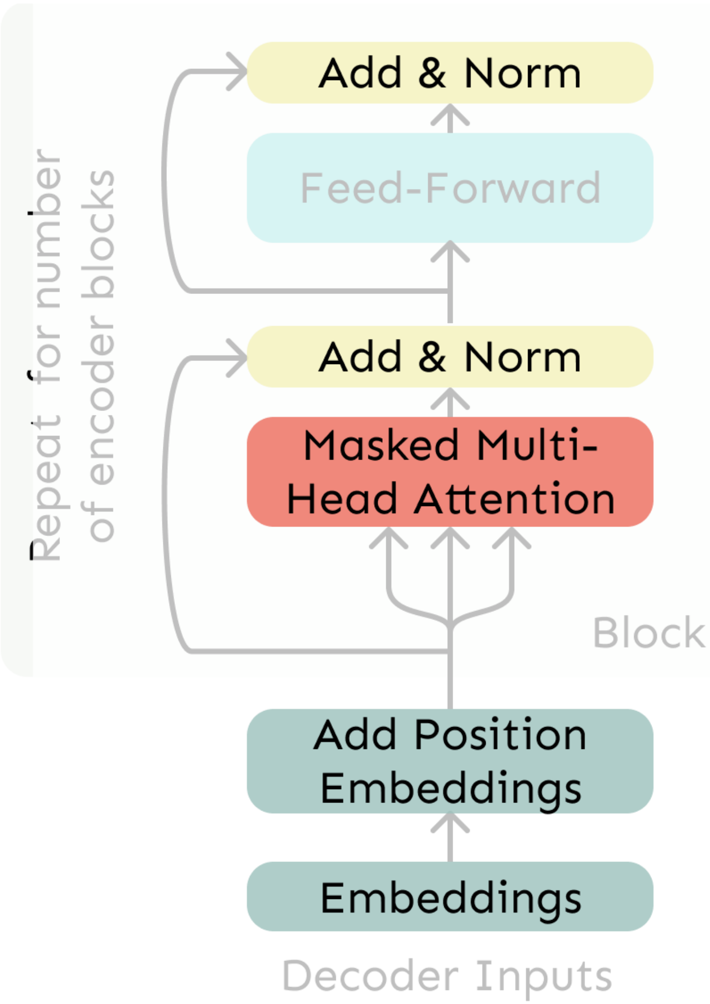
$$\mathrm{output}_\ell = \mathrm{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$$

- We divide the attention scores by $\sqrt{d/h}$, to stop the scores from becoming large just as a function of d/h (The dimensionality divided by the number of heads.)

$$\mathrm{output}_\ell = \mathrm{softmax}\left(\frac{XQ_\ell K_\ell^\top X^\top}{\sqrt{d/h}}\right) * XV_\ell$$

The Transformer Decoder

- Now that we've replaced self-attention with multi-head self-attention, we'll go through two **optimization tricks** that end up being :
    - **Residual Connections**
    - **Layer Normalization**
- In most Transformer diagrams, these are often written together as "Add & Norm"

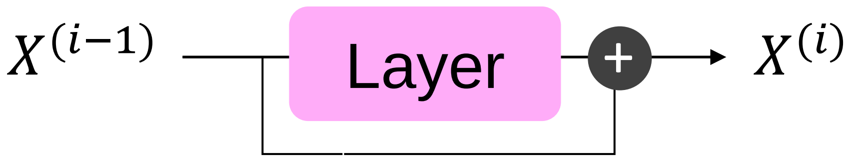


Transformer Decoder

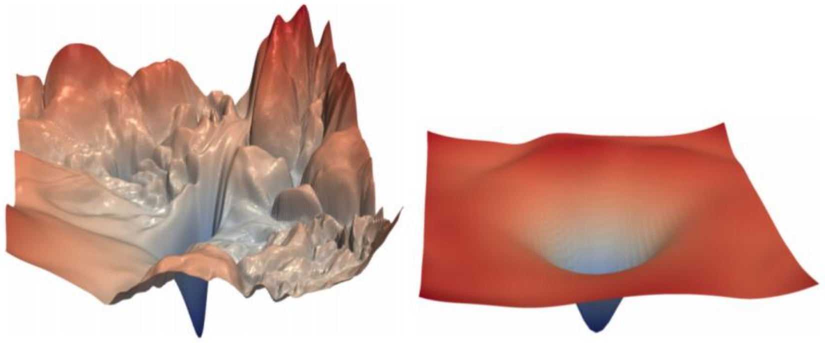
The Transformer Encoder: **Residual connections** [He et al., 2016]

- **Residual connections** are a trick to help models train better.
  - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)

  - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn "the residual" from the previous layer)



  - Gradient is **great** through the residual connection; it's 1!
  - Bias towards the identity function!



[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

The Transformer Encoder: **Layer normalization** [Ba et al., 2016]

- **Layer normalization** is a trick to help models train faster.
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation **within each layer**.
  - LayerNorm's success may be due to its normalizing gradients [Xu et al., 2019]
- Let $x \in \mathbb{R}^d$ be an individual (word) vector in the model.
- Let $\mu = \sum_{j=1}^{d} x_j$; this is the mean; $\mu \in \mathbb{R}$.
- Let $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^{d} \left(x_j - \mu\right)^2}$; this is the standard deviation; $\sigma \in \mathbb{R}$.
- Let $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ be learned "gain" and "bias" parameters. (Can omit!)
- Then layer normalization computes:

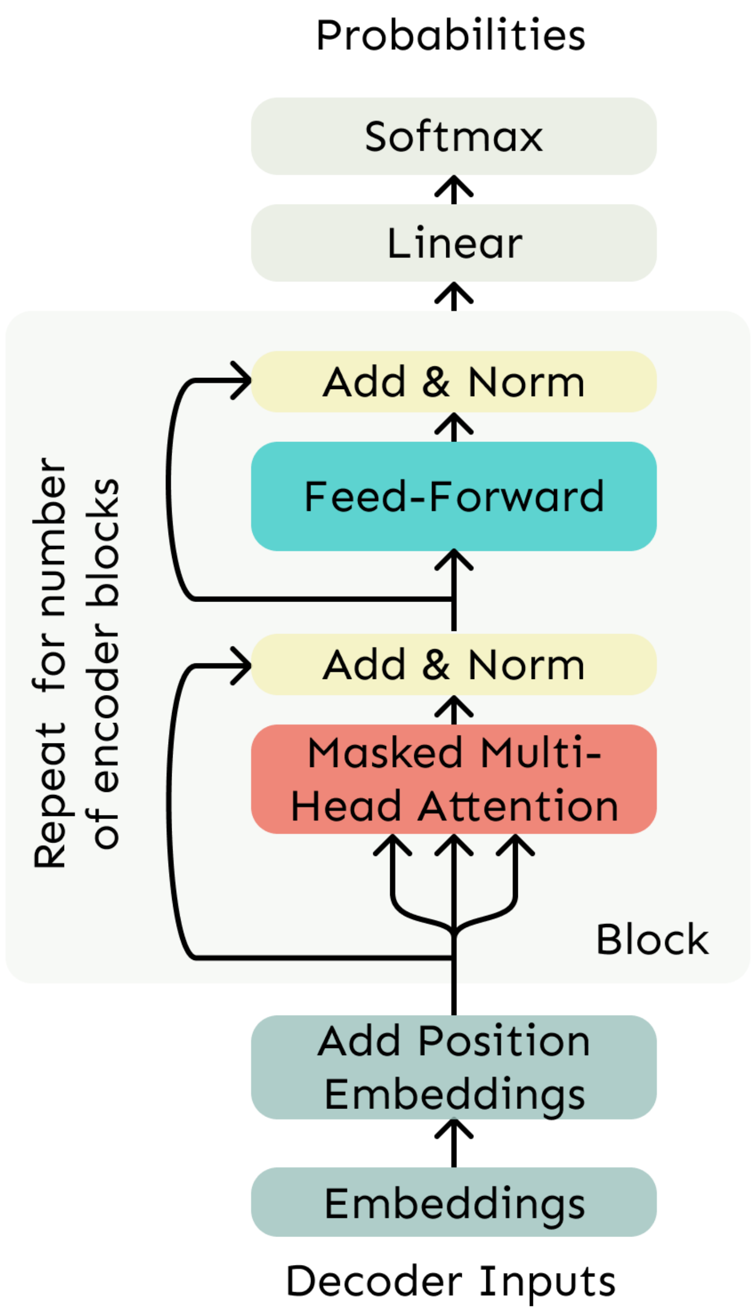
$$\text{output} = \frac{x - \mu}{\sqrt{\sigma} + \epsilon} * \gamma + \beta$$

Normalize by scalar mean and variance

Modulate by learned elementwise gain and bias

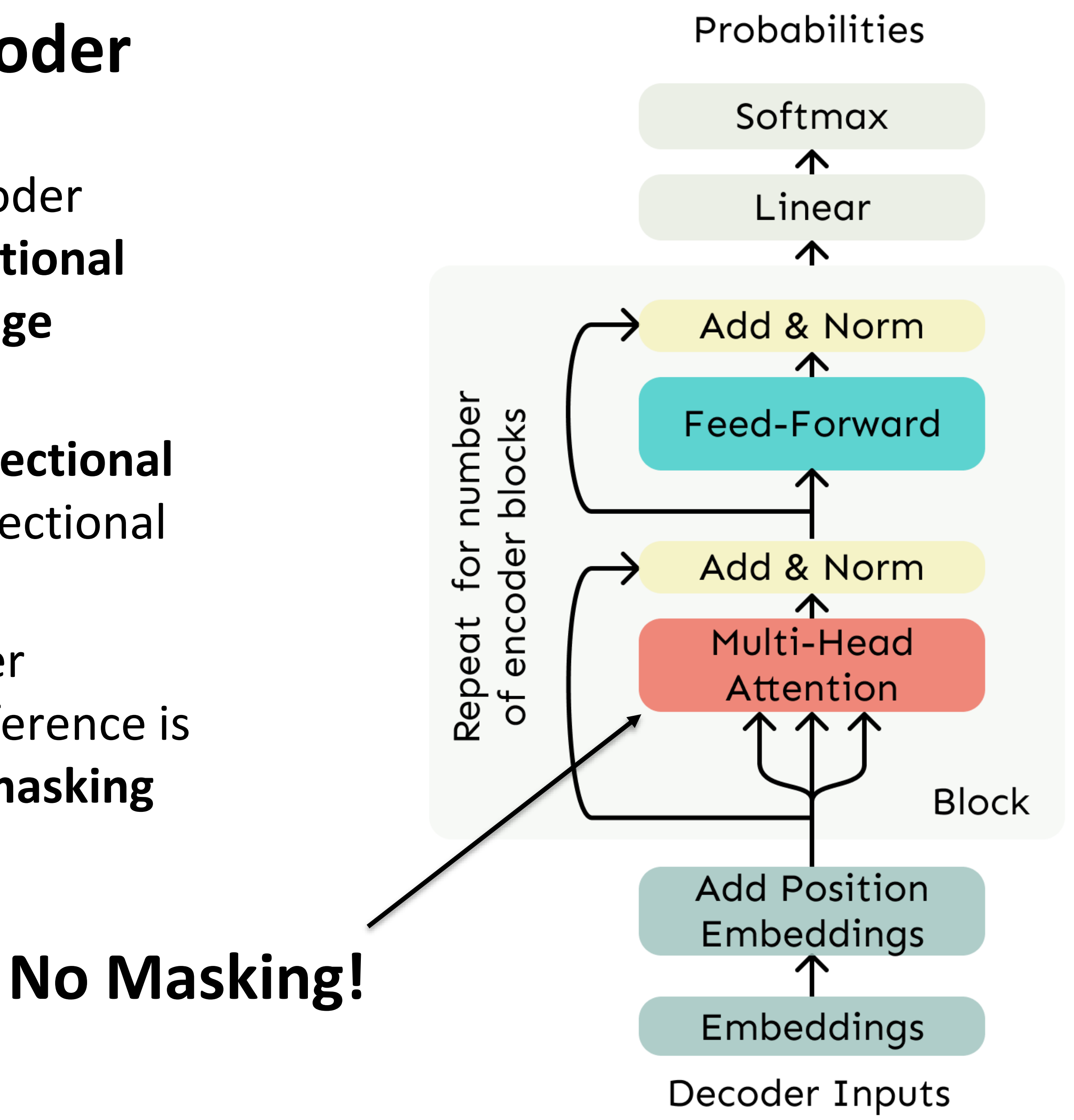
The Transformer Decoder

- The Transformer Decoder is a stack of Transformer Decoder **Blocks**.
- Each Block consists of:
  - Self-attention
  - Add & Norm
  - Feed-Forward
  - Add & Norm
- That’s it! We’ve gone through the Transformer Decoder.



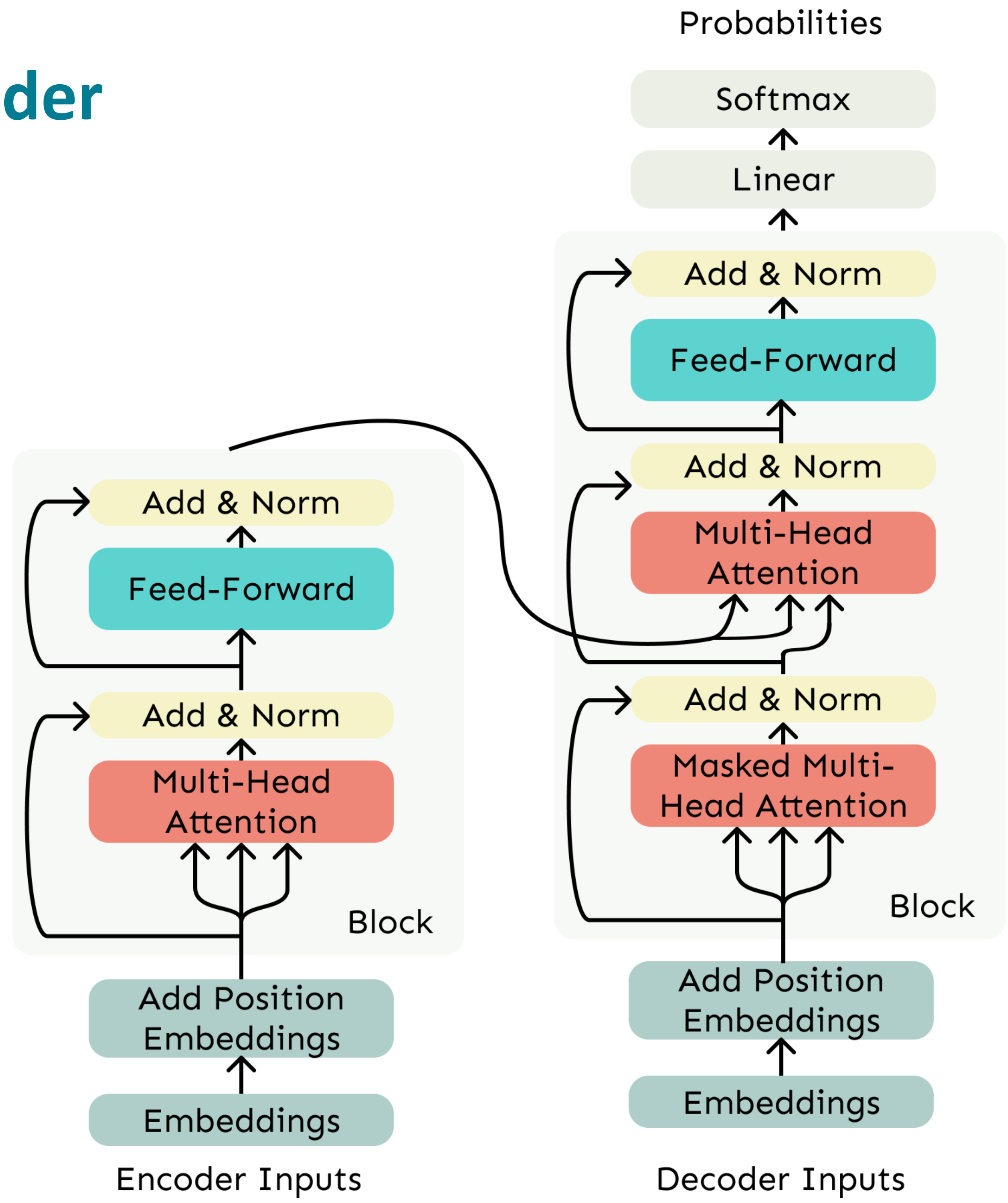
The Transformer Encoder

- The Transformer Decoder constrains to **unidirectional context**, as for **language models.**
- What if we want **bidirectional context**, like in a bidirectional RNN?
- This is the Transformer Encoder. The only difference is that we **remove the masking** in the self-attention.



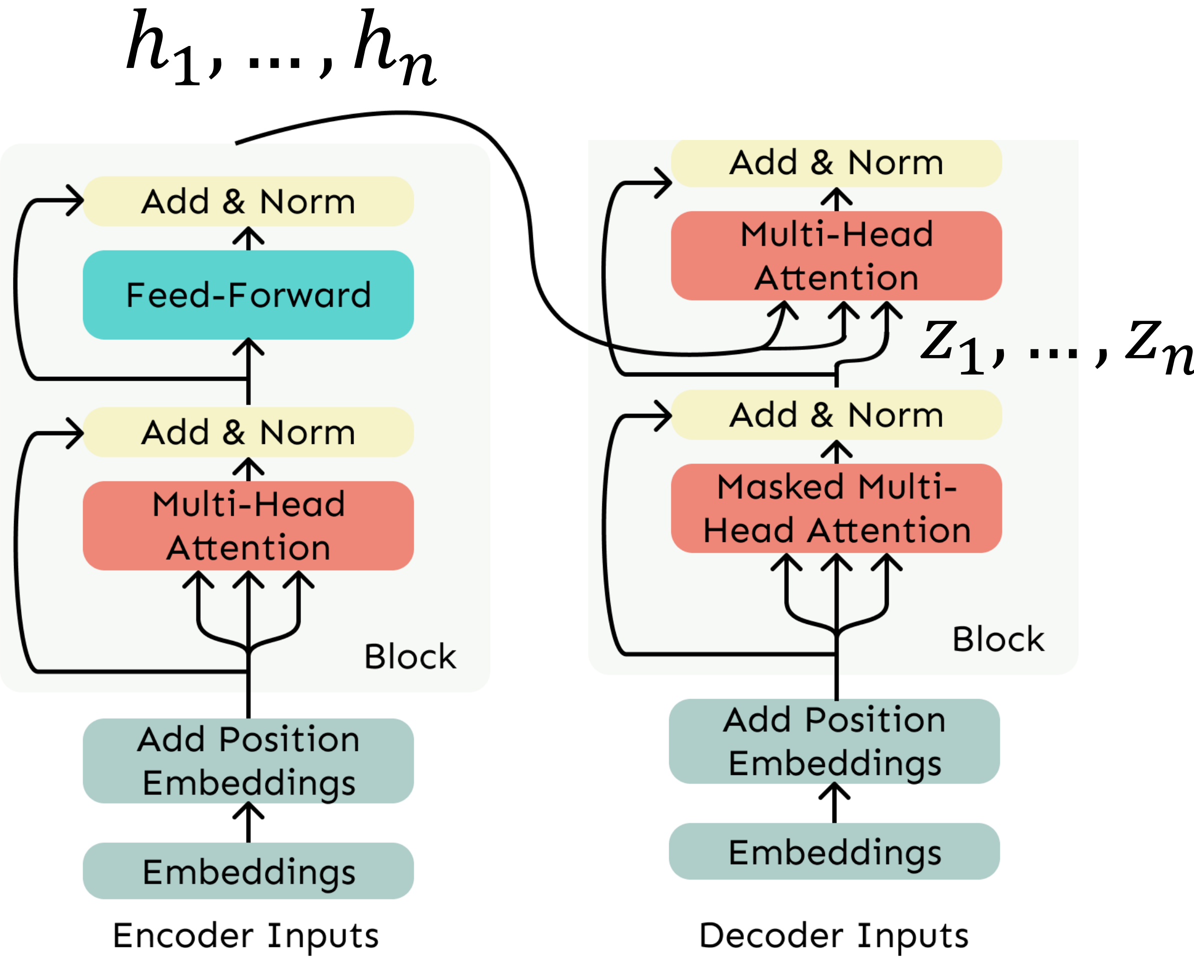
The Transformer Encoder-Decoder

- Recall that in machine translation, we processed the source sentence with a **bidirectional** model and generated the target with a **unidirectional model**.
- For this kind of seq2seq format, we often use a Transformer Encoder-Decoder.
- We use a normal Transformer Encoder.
- Our Transformer Decoder is modified to perform **cross-attention** to the output of the Encoder.



Cross-attention (details)

- We saw that self-attention is when keys, queries, and values come from the same source.
- In the decoder, we have attention that looks more like what we saw last week.
- Let $h_1, \dots, h_n$ be **output** vectors **from** the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let $z_1, \dots, z_n$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the **encoder** (like a memory):
  - $k_i = Kh_i, v_i = Vh_i.$
- And the queries are drawn from the **decoder**, $q_i = Qz_i$.



Great Results with Transformers: Machine Translation

First, Machine Translation results from the original Transformers paper!

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

[Test sets: WMT 2014 English-German and English-French]

[Vaswani et al., 2017]

Great Results with Transformers: SuperGLUE

SuperGLUE is a suite of challenging NLP tasks, including question-answering, word sense disambiguation, coreference resolution, and natural language inference.

	Rank	Name	Model	URL	Score	BoolQ	CB	COPA	MultiRC	ReCoRD	RTE	WiC	WSC	AX-b	AX-g
	1	JDExplore d-team	Vega v2		91.3	90.5	98.6/99.2	99.4	88.2/62.4	94.4/93.9	96.0	77.4	98.6	-0.4	100.0/50.0
+	2	Liam Fedus	ST-MoE-32B		91.2	92.4	96.9/98.0	99.2	89.6/65.8	95.1/94.4	93.5	77.7	96.6	72.3	96.1/94.1
	3	Microsoft Alexander v-team	Turing NLR v5		90.9	92.0	95.9/97.6	98.2	88.4/63.0	96.4/95.9	94.1	77.1	97.3	67.8	93.3/95.5
	4	ERNIE Team - Baidu	ERNIE 3.0		90.6	91.0	98.6/99.2	97.4	88.6/63.2	94.7/94.2	92.6	77.4	97.3	68.6	92.7/94.7
	5	Yi Tay	PaLM 540B		90.4	91.9	94.4/96.0	99.0	88.7/63.6	94.2/93.3	94.1	77.4	95.9	72.9	95.5/90.4
+	6	Zirui Wang	T5 + UDG, Single Model (Google Brain)		90.4	91.4	95.8/97.6	98.0	88.3/63.0	94.2/93.5	93.0	77.9	96.6	69.1	92.7/91.9
+	7	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.3	90.4	95.7/97.6	98.4	88.2/63.7	94.5/94.1	93.2	77.5	95.9	66.7	93.3/93.8
	8	SuperGLUE Human Baselines	SuperGLUE Human Baselines		89.8	89.0	95.8/98.9	100.0	81.8/51.9	91.7/91.3	93.6	80.0	100.0	76.6	99.3/99.7
+	9	T5 Team - Google	T5		89.3	91.2	93.9/96.8	94.8	88.1/63.3	94.1/93.4	92.5	76.9	93.8	65.6	92.7/91.9
	10	SPoT Team - Google	Frozen T5 1.1 + SPoT		89.2	91.1	95.8/97.6	95.6	87.9/61.9	93.3/92.4	92.9	75.8	93.8	66.9	83.1/82.6

[Test sets: SuperGLUE Leaderboard Version: 2.0]

[Wang et al., 2019]

Great Results with Transformers: Rise of Large Language Models!

Today, Transformer-based models dominate LMSYS Chatbot Arena Leaderboard!

Rank	🤖 Model	⭐ Arena Elo	📊 95% CI	🗳️ Votes	Organization	License	Knowledge Cutoff
1	GPT-4-Turbo-2024-04-09	1258	+4/-4	26444	OpenAI	Proprietary	2023/12
1	GPT-4-1106-preview	1253	+3/-3	68353	OpenAI	Proprietary	2023/4
1	Claude 3 Opus	1251	+3/-3	71500	Anthropic	Proprietary	2023/8
2	Gemini 1.5 Pro API-0409-Preview	1249	+4/-5	22211	Google	Proprietary	2023/11
3	GPT-4-0125-preview	1248	+2/-3	58959	OpenAI	Proprietary	2023/12
6	Meta Llama 3 70b Instruct	1213	+4/-6	15809	Meta	Llama 3 Community	2023/12
6	Bard (Gemini Pro)	1208	+7/-6	12435	Google	Proprietary	Online
7	Claude 3 Sonnet	1201	+4/-2	73414	Anthropic	Proprietary	2023/8

Gemini / Bard (Google)

ChatGPT / GPT-4 (OpenAI)

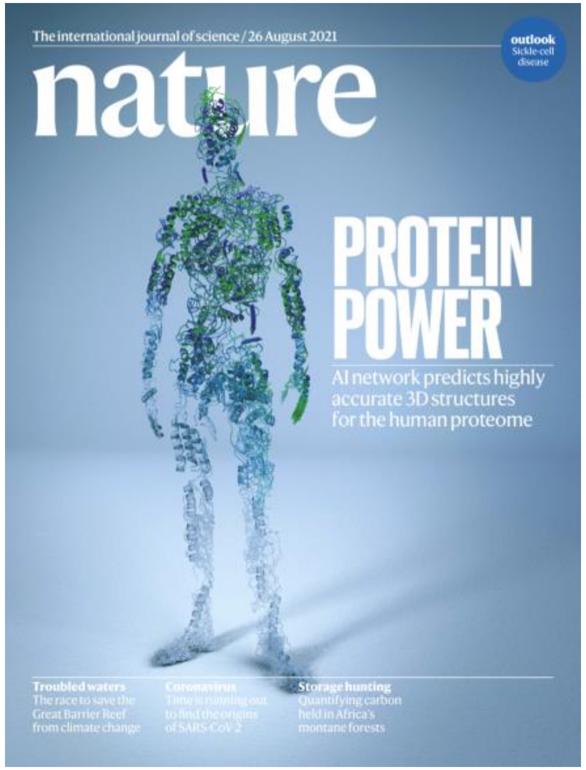
Claude 3 (Anthropic)

Llama 3 (Meta)

[Chiang et al., 2024]

Transformers Even Show Promise Outside of NLP

Protein Folding



[Jumper et al. 2021] aka AlphaFold2!

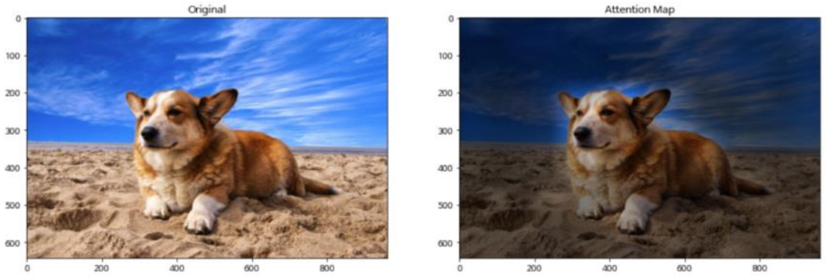
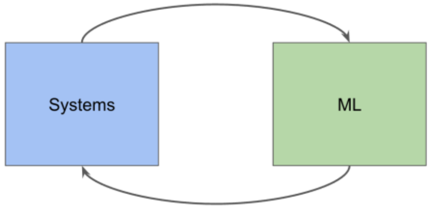
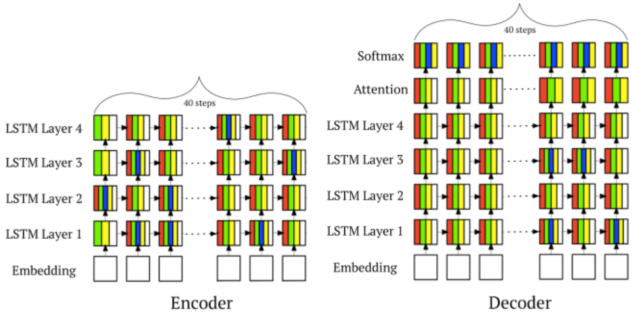


Image Classification

[Dosovitskiy et al. 2020]: Vision Transformer (ViT) outperforms ResNet-based baselines with substantially less compute.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k





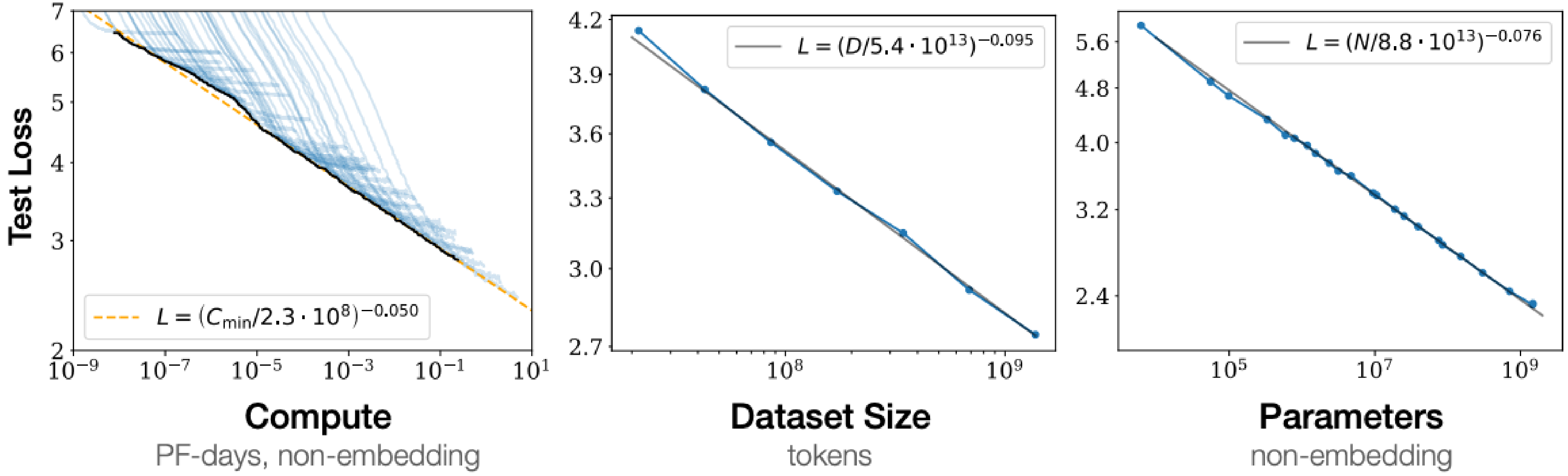
ML for Systems

[Zhou et al. 2020]: A Transformer-based compiler model (GO-one) speeds up a Transformer model!

Model (#devices)	**GO-one** (s)	HP (s)	METIS (s)	HDP (s)	Run time speed up over HP / HDP	Search speed up over HDP
2-layer RNNLM (2)	0.173	0.192	0.355	0.191	9.9% / 9.4%	2.95x
4-layer RNNLM (4)	0.210	0.239	0.503	0.251	13.8% / 16.3%	1.76x
8-layer RNNLM (8)	0.320	0.332	OOM	0.764	3.8% / 58.1%	27.8x
2-layer GNMT (2)	0.301	0.384	0.344	0.327	27.6% / 14.3%	30x
4-layer GNMT (4)	0.350	0.469	0.466	0.432	34% / 23.4%	58.8x
[illegible]	0.440	0.562	OOM	0.693	21.7% / 36.5%	7.35x
2-layer Transformer-XL (2)	0.223	0.268	0.37	0.262	20.1% / 17.4%	40x
4-layer Transformer-XL (4)	0.230	0.27	OOM	0.259	17.4% / 12.6%	26.7x
8-layer Transformer-XL (8)	0.350	0.46	OOM	0.425	23.9% / 16.7%	16.7x
[illegible]	0.229	0.312	OOM	0.301	26.6% / 23.9%	13.5x
Inception (2) b64	0.423	0.731	OOM	0.498	42.1% / 29.3%	21.0x
AmoebaNet (4)	0.394	0.44	0.426	0.418	26.1% / 6.1%	58.8x
2-stack 18-layer WaveNet (2)	0.317	0.376	OOM	0.354	18.6% / 11.7%	6.67x
4-stack 36-layer WaveNet (4)	0.659	0.988	OOM	0.721	50% / 9.4%	20x
GEOMEAN	-	-	-	-	**20.5% / 18.2%**	**15x**

Scaling Laws: Are Transformers All We Need?

- With Transformers, language modeling performance improves smoothly as we increase model size, training data, and compute resources in tandem.
- This power-law relationship has been observed over multiple orders of magnitude with no sign of slowing!
- If we keep scaling up these models (with no change to the architecture), could they eventually match or exceed human-level performance?



[Kaplan et al., 2020]

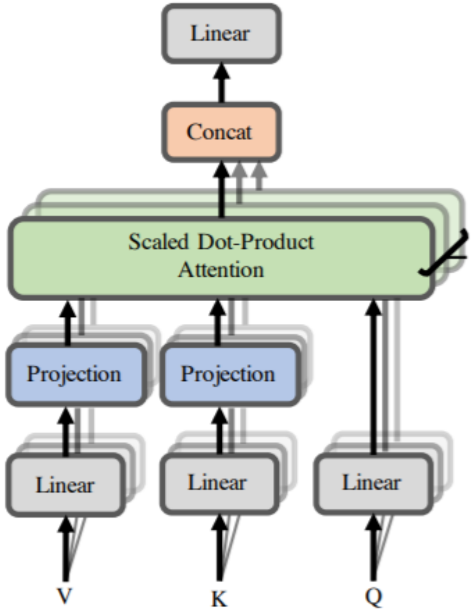
What would we like to fix about the Transformer?

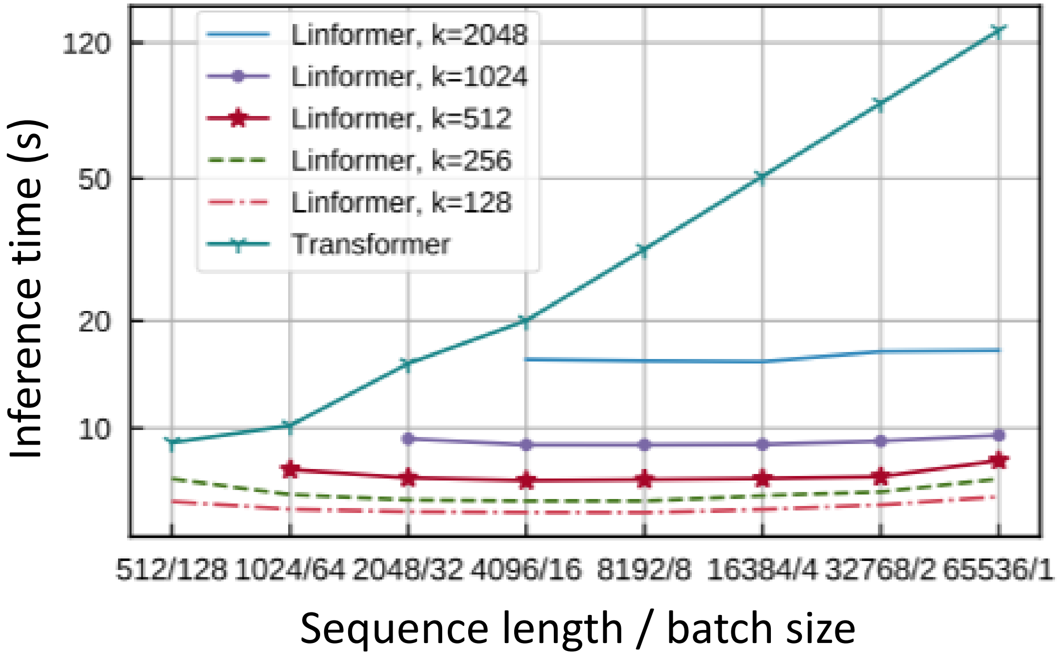
- **Quadratic compute in self-attention (today)**:
  - Computing all pairs of interactions means our computation grows **quadratically** with the sequence length!
  - For recurrent models, it only grew linearly!
- **Position representations**:
  - Are simple absolute indices the best we can do to represent position?
  - As we learned: Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]
  - Rotary Embeddings [Su et al., 2021]

Recent work on improving on quadratic self-attention cost

- Considerable recent work has gone into the question, *Can we build models like Transformers without paying the* $O(T^2)$ *all-pairs self-attention cost?*
- For example, **Linformer** [Wang et al., 2020]

Key idea: map the sequence length dimension to a lower-dimensional space for values, keys

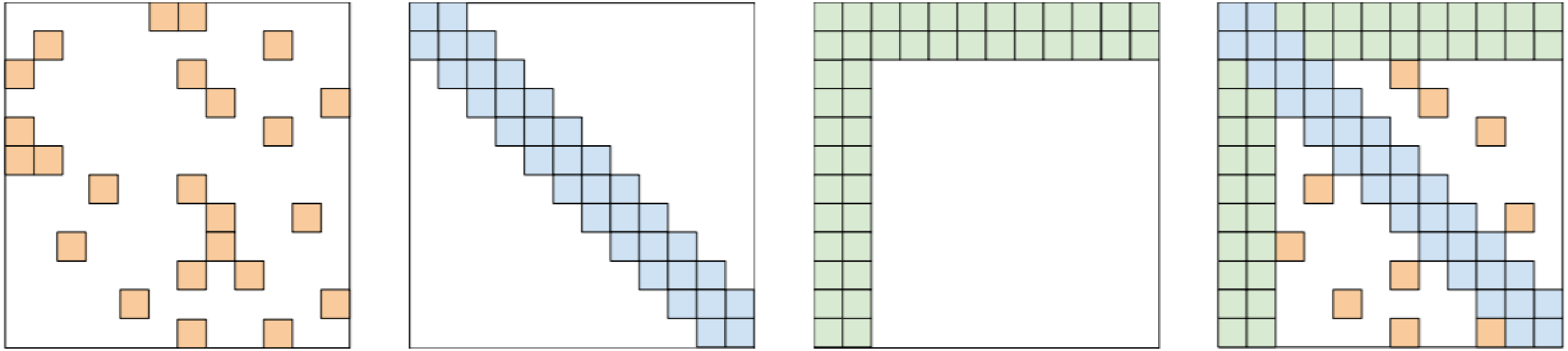




Recent work on improving on quadratic self-attention cost

- Considerable recent work has gone into the question, *Can we build models like Transformers without paying the $O(T^2)$ all-pairs self-attention cost?*
- For example, **BigBird** [Zaheer et al., 2021]

Key idea: replace all-pairs interactions with a family of other interactions, **like local windows**, **looking at everything**, and **random interactions**.



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Do Transformer Modifications Transfer?

- "Surprisingly, we find that most modifications do not meaningfully improve performance."

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
GeLU	$223M$	$11.1T$	3.58	2.179 ± 0.003	1.838	**75.79**	**17.86**	**25.13**	26.47
Swish	$223M$	$11.1T$	3.62	2.186 ± 0.003	1.847	**73.77**	17.74	**24.34**	**26.75**
ELU	$223M$	$11.1T$	3.56	2.270 ± 0.007	1.932	67.83	16.73	23.02	26.08
GLU	$223M$	$11.1T$	3.59	2.174 ± 0.003	**1.814**	**74.20**	**17.42**	24.34	**27.12**
GeGLU	$223M$	$11.1T$	3.55	2.130 ± 0.006	**1.792**	**75.96**	**18.27**	**24.87**	**26.87**
ReGLU	$223M$	$11.1T$	3.57	2.145 ± 0.004	**1.803**	**76.17**	**18.36**	**24.87**	**27.02**
SeLU	$223M$	$11.1T$	3.55	2.315 ± 0.004	1.948	68.76	16.76	22.75	25.99
SwiGLU	$223M$	$11.1T$	3.53	2.127 ± 0.003	**1.789**	**76.00**	**18.20**	**24.34**	**27.02**
LiGLU	$223M$	$11.1T$	3.59	2.149 ± 0.005	**1.798**	**75.34**	**17.97**	**24.34**	26.53
Sigmoid	$223M$	$11.1T$	3.63	2.291 ± 0.019	1.867	**74.31**	17.51	23.02	26.30
Softplus	$223M$	$11.1T$	3.47	2.207 ± 0.011	1.850	**72.45**	17.65	**24.34**	**26.89**
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
24 layers, $d_{ff} = 1536$, $H = 6$	$224M$	$11.1T$	3.33	2.200 ± 0.007	1.843	**74.89**	17.75	**25.13**	**26.89**
18 layers, $d_{ff} = 2048$, $H = 8$	$223M$	$11.1T$	3.38	2.185 ± 0.005	**1.831**	**76.45**	16.83	**24.34**	**27.10**
8 layers, $d_{ff} = 4608$, $H = 18$	$223M$	$11.1T$	3.69	2.190 ± 0.005	1.847	**74.58**	17.69	**23.28**	**26.85**
6 layers, $d_{ff} = 6144$, $H = 24$	$223M$	$11.1T$	3.70	2.201 ± 0.010	1.857	**73.55**	17.59	**24.60**	**26.66**
Block sharing	$65M$	$11.1T$	3.91	2.497 ± 0.037	2.164	64.50	14.53	21.96	25.48
+ Factorized embeddings	$45M$	$9.4T$	4.21	2.631 ± 0.305	2.183	60.84	14.00	19.84	25.27
+ Factorized & shared embeddings	$20M$	$9.1T$	4.37	2.907 ± 0.313	2.385	53.95	11.37	19.84	25.19
Encoder only block sharing	$170M$	$11.1T$	3.68	2.298 ± 0.023	1.929	69.60	16.23	23.02	26.23
Decoder only block sharing	$144M$	$11.1T$	3.70	2.352 ± 0.029	2.082	67.93	16.13	**23.81**	26.08
Factorized Embedding	$227M$	$9.4T$	3.80	2.208 ± 0.006	1.855	70.41	15.92	22.75	26.50
Factorized & shared embeddings	$202M$	$9.1T$	3.92	2.320 ± 0.010	1.952	68.69	16.33	22.22	26.44
Tied encoder/decoder input embeddings	$248M$	$11.1T$	3.55	2.192 ± 0.002	1.840	**71.70**	17.72	**24.34**	26.49
Tied decoder input and output embeddings	$248M$	$11.1T$	3.57	2.187 ± 0.007	**1.827**	**74.86**	17.74	**24.87**	**26.67**
Untied embeddings	$273M$	$11.1T$	3.53	2.195 ± 0.005	**1.834**	**72.99**	17.58	**23.28**	26.48
Adaptive input embeddings	$204M$	$9.2T$	3.55	2.250 ± 0.002	1.899	66.57	16.21	**24.07**	**26.66**
Adaptive softmax	$204M$	$9.2T$	3.60	2.364 ± 0.005	1.982	**72.91**	16.67	21.16	25.56
Adaptive softmax without projection	$223M$	$10.8T$	3.43	2.229 ± 0.009	1.914	**71.82**	17.10	23.02	25.72
Mixture of softmaxes	$232M$	$16.3T$	2.24	2.227 ± 0.017	**1.821**	**76.77**	17.62	22.75	**26.82**
Transparent attention	$223M$	$11.1T$	3.33	2.181 ± 0.014	1.874	54.31	10.40	21.16	**26.80**
Dynamic convolution	$257M$	$11.8T$	2.65	2.403 ± 0.009	2.047	58.30	12.67	21.16	17.03
Lightweight convolution	$224M$	$10.4T$	4.07	2.370 ± 0.010	1.989	63.07	14.86	23.02	24.73
Evolved Transformer	$217M$	$9.9T$	3.09	2.220 ± 0.003	1.863	**73.67**	10.76	**24.07**	26.58
Synthesizer (dense)	$224M$	$11.4T$	3.47	2.334 ± 0.021	1.962	61.03	14.27	16.14	**26.63**
Synthesizer (dense plus)	$243M$	$12.6T$	3.22	2.191 ± 0.010	1.840	**73.98**	16.96	**23.81**	**26.71**
Synthesizer (dense plus alpha)	$243M$	$12.6T$	3.01	2.180 ± 0.007	**1.828**	**74.25**	17.02	**23.28**	26.61
Synthesizer (factorized)	$207M$	$10.1T$	3.94	2.341 ± 0.017	1.968	62.78	15.39	**23.55**	26.42
Synthesizer (random)	$254M$	$10.1T$	4.08	2.326 ± 0.012	2.009	54.27	10.35	19.56	26.44
Synthesizer (random plus)	$292M$	$12.0T$	3.63	2.189 ± 0.004	1.842	**73.32**	17.04	**24.87**	26.43
Synthesizer (random plus alpha)	$292M$	$12.0T$	3.42	2.186 ± 0.007	**1.828**	**75.24**	17.08	**24.08**	26.39
Universal Transformer	$84M$	$40.0T$	0.88	2.406 ± 0.036	2.053	70.13	14.09	19.05	23.91
Mixture of experts	$648M$	$11.7T$	3.20	2.148 ± 0.006	**1.785**	**74.55**	**18.13**	**24.08**	**26.94**
Switch Transformer	$1100M$	$11.7T$	3.18	2.135 ± 0.007	**1.758**	**75.38**	**18.02**	**26.19**	**26.81**
Funnel Transformer	$223M$	$1.9T$	4.30	2.288 ± 0.008	1.918	67.34	16.26	22.75	23.20
Weighted Transformer	$280M$	$71.0T$	0.59	2.378 ± 0.021	1.989	69.04	16.98	23.02	26.30
Product key memory	$421M$	$386.6T$	0.25	2.155 ± 0.003	**1.798**	**75.16**	17.04	**23.55**	**26.73**

Do Transformer Modifications Transfer Across Implementations and Applications?

Sharan Narang* Hyung Won Chung Yi Tay William Fedus
Thibault Fevry† Michael Matena† Karishma Malkan† Noah Fiedel
Noam Shazeer Zhenzhong Lan† Yanqi Zhou Wei Li
Nan Ding Jake Marcus Adam Roberts Colin Raffel†