Transformers

Aaron Mueller
CS599 B1: Advanced Natural Language Processing
Sep. 8, 2026
Fall 2026

Motivation

- Feedforward neural language models can leverage the meaning of words, but can’t handle long contexts effectively

- Smarter architectures like recurrent neural networks can do this, but the current state-of-the-art is **Transformers**, which use **self-attention**

Intuition

- You can think of self-attention as doing two things:
    1. Figuring out which tokens relate to each other
    2. Figuring out what information to move between token positions

- Transformers use the following components, each of which we'll discuss in detail:
    - **Multi-head self-attention**
    - **Causal masking**
    - **Feed-forward layers with non-linearities**
    - **LayerNorm**
    - **Positional embeddings** (in Thursday's lecture)

Example

Long-context Question Answering

Björk

94 languages

Article Talk Read Edit View history

From Wikipedia, the free encyclopedia

This article is about the singer. For her self-titled album, see Björk (album). For other uses, see Björk (disambiguation).
This is an Icelandic name. The last name is patronymic, not a family name; this person is referred to by the first name Björk.

Björk Guðmundsdóttir OTF (/bjɜːrk/ *BYURK*, Icelandic: [pjœr̥k ˈkvʏðmʏntsˌtouhtɪr̥] ⓘ; born 21 November 1965) is an Icelandic singer, songwriter, composer, record producer and actress. Noted for her distinct voice, three-octave vocal range, and eccentric public persona, she has developed an eclectic musical style over a career spanning five decades, drawing on electronica, pop, dance, trip hop, jazz, and avant-garde music. She is regarded as one of the most influential musicians of her era.[2]

Born and raised in Reykjavík, Björk began her music career at the age of 11 and gained international recognition as the lead singer of the alternative rock band the Sugarcubes by the age of 21.[3] After the Sugarcubes disbanded in 1992, Björk gained prominence as a solo artist with her albums *Debut* (1993), *Post* (1995), and

Björk
OTF

Björk performing in Paris during her Björk Orkestral show in 2022

Born Björk Guðmundsdóttir
21 November 1965 (age 60)
Reykjavík, Iceland

Feed all of this into our model, then ask:
“What is the name of Björk’s second album?”

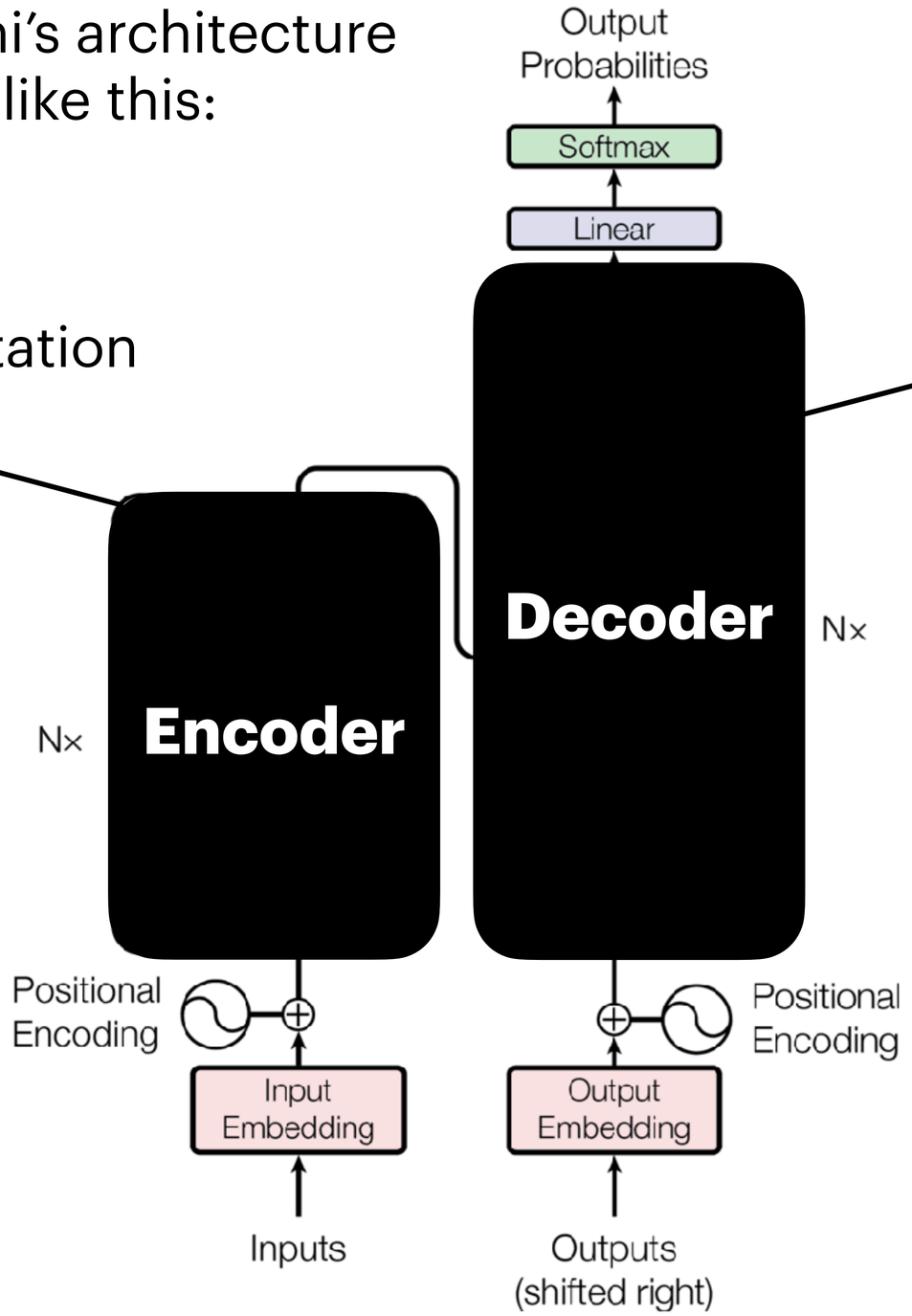
Recurrent and feedforward architectures struggle with this; they forget information from many positions in the past

Attention allows us to access info arbitrarily far back in our context

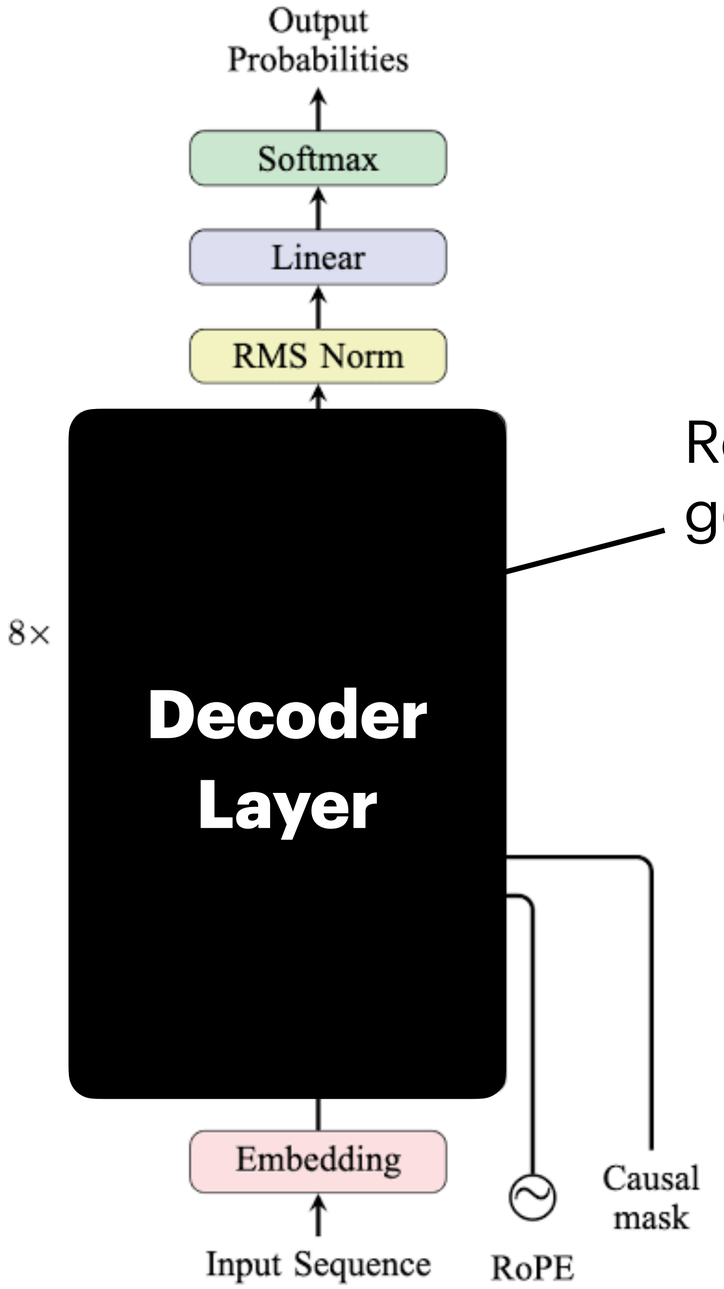
Vaswani's architecture looked like this:

Reads the left *and* right context, yields representation of current token

Reads only the left context; generates the next token



In this class, our architecture will look more like this:

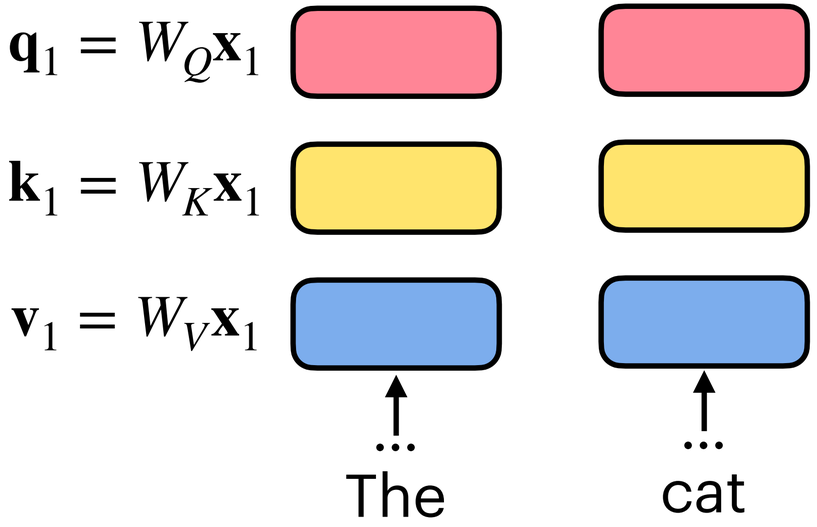


Reads only the left context; generates the next token

Self-attention

1. Derive three vectors from hidden state $\mathbf{x}_1$: the **query**, **key**, and **value**

Intuition: Think of retrieval. We have a **query** (input), which is compared to all **keys**. We retrieve the **value**(s) associated with the **key**(s) most similar to the **query**.



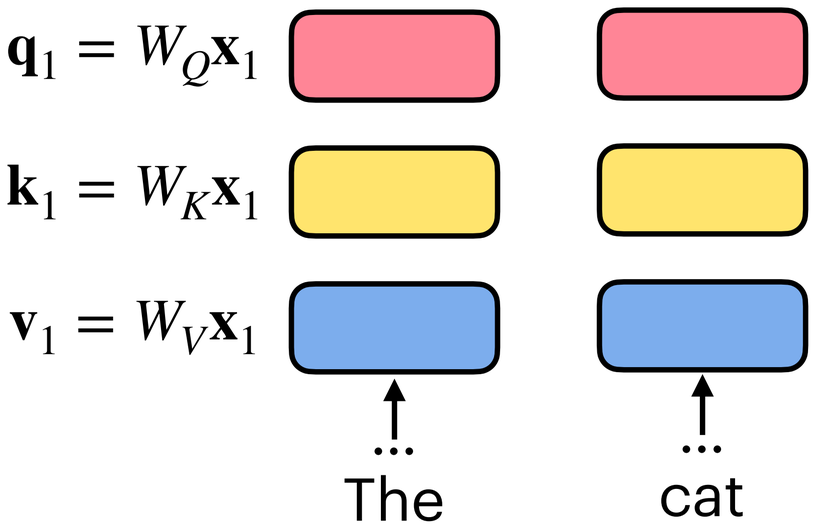
2. For all i, j, dot query of $\mathbf{x}_i$ with key of $\mathbf{x}_j$:

Attention score → $A_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$

3. Divide by square root of the dimensionality of $\mathbf{k}$, then softmax:

Attention weight → $\alpha_{ij} = \text{softmax}\left(\frac{A_{ij}}{\sqrt{d_\mathbf{k}}}\right)$

Self-attention



2. For all i, j, dot **query** of $\mathbf{x}_i$ with **key** of $\mathbf{x}_j$:

$$A_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$$

3. Divide by square root of the dimensionality of $\mathbf{k}$ and softmax:

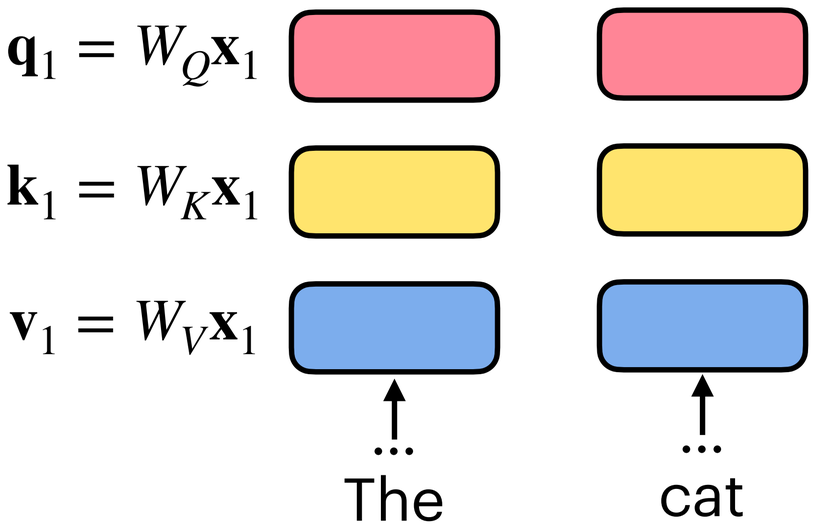
$$\alpha_{ij} = \text{Softmax}\left(\frac{A_{ij}}{\sqrt{d_{\mathbf{k}}}}\right)$$

4. For all j, multiply α_{ij} by **value** $\mathbf{v}_j$ and sum:

$$\mathbf{z}_i = \sum_{j=1}^{n} \alpha_{ij}\mathbf{v}_j$$

This is the output of the self-attention layer for token $\mathbf{x}_i$.

Self-attention



2. For all i, j, dot **query** of $\mathbf{x}_i$ with **key** of $\mathbf{x}_j$:

$$A_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$$

3. Divide by square root of the dimensionality of $\mathbf{k}$ and softmax:

$$\alpha_{ij} = \text{Softmax}\left(\frac{A_{ij}}{\sqrt{d_{\mathbf{k}}}}\right)$$

(You'll sometimes see this referred to as "scaled dot product attention")

4. For all j, multiply α_{ij} by **value** $\mathbf{v}_j$ and sum:

$$\mathbf{z}_i = \sum_{j=1}^{n} \alpha_{ij}\mathbf{v}_j$$

This is the output of the self-attention layer for token $\mathbf{x}_i$.

Example

Long-context Question Answering

Björk

94 languages

Article Talk | Read Edit View history

From Wikipedia, the free encyclopedia

This article is about the singer. For her self-titled album, see Björk (album). For other uses, see Björk (disambiguation).
This is an Icelandic name. The last name is patronymic, not a family name; this person is referred to by the first name Björk.

Björk Guðmundsdóttir OTF (/bjɜːrk/ *BYURK*, Icelandic: [pjœr̥k ˈkvʏðmʏnts̥touhtɪr̥]; born 21 November 1965) is an Icelandic singer, songwriter, composer, record producer and actress. Noted for her distinct voice, three-octave vocal range, and eccentric public persona, she has developed an eclectic musical style over a career spanning five decades, drawing on electronica, pop, dance, trip hop, jazz, and avant-garde music. She is regarded as one of the most influential musicians of her era.[2]

Born and raised in Reykjavík, Björk began her music career at the age of 11 and gained international recognition as the lead singer of the alternative rock band the Sugarcubes by the age of 21.[3] After the Sugarcubes disbanded in 1992, Björk gained prominence as a solo artist with her albums *Debut* (1993), *Post* (1995), and

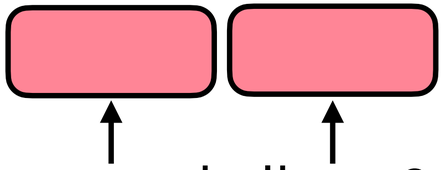
Björk
OTF

Björk performing in Paris during her Björk Orkestral show in 2022

Born Björk Guðmundsdóttir
21 November 1965 (age 60)
Reykjavík, Iceland

$$\mathbf{q}_{t-1} = W_Q \mathbf{x}_{t-1}$$

$$\mathbf{q}_{t-2} = W_Q \mathbf{x}_{t-2}$$



What is the name of Bjork's second album?

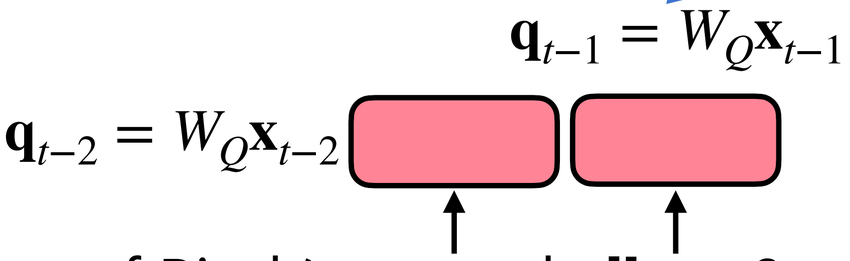
Compute the **key** of *all* tokens in the article (and the question)

If query and key have high dot product, bring the values from the key positions to the query positions so they can be used to help us answer the question

Example

Long-context Question Answering

of 21.[3] After the Sugarcubes disbanded in 1992, Björk gained prominence as a solo artist with her albums *Debut* (1993), *Post* (1995), and



What is the name of Bjork's second **album**?

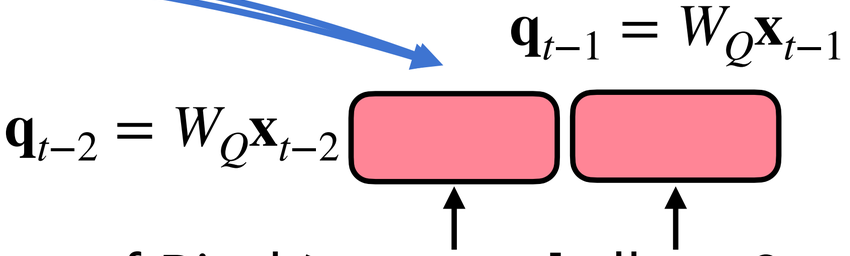
Compute the **key** of *all* tokens in the article (and the question)

If query and key have high dot product, bring the values from the key positions to the query positions so they can be used to help us answer the question

Example

Long-context Question Answering

of 21.[3] After the Sugarcubes disbanded in 1992, Björk gained prominence as a solo artist with her albums *Debut* (1993), *Post* (1995), and



What is the name of Bjork's **second** album?

Compute the **key** of *all* tokens in the article (and the question)

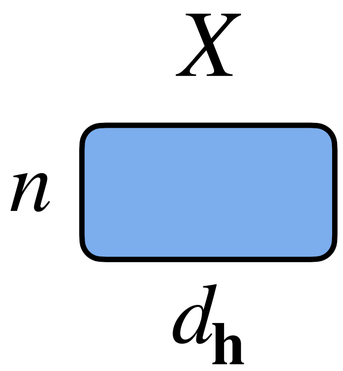
If query and key have high dot product, bring the **values** from the key positions to the query positions so they can be used to help us answer the question

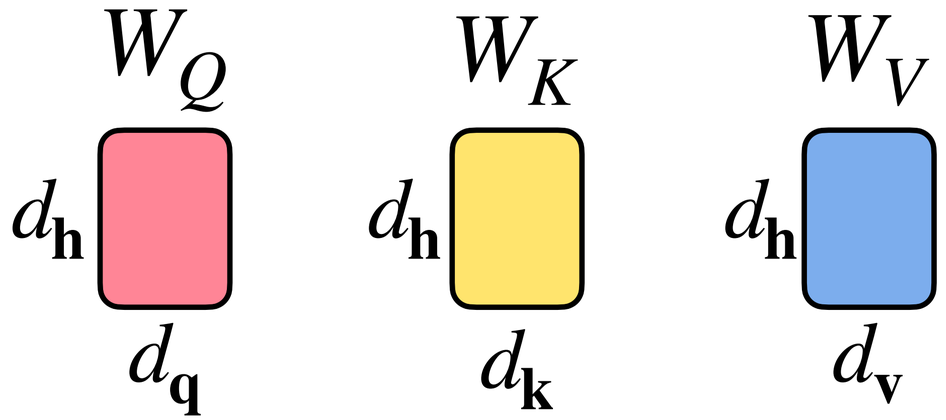
Self-attention

Batching attention computations

Doing this token-by-token would be very slow. In practice, we do all of this in parallel for all token positions. Let's do this in matrix form:

$$X = [\mathbf{x}_1; \mathbf{x}_2; \ldots; \mathbf{x}_n]$$





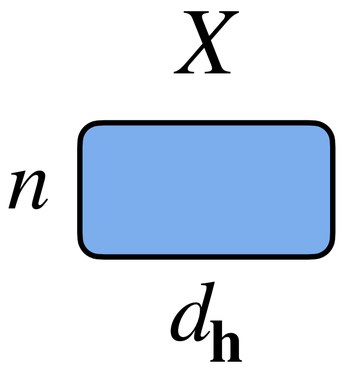
$$Q = XW_Q \qquad V = XW_V$$

$$K = XW_K$$

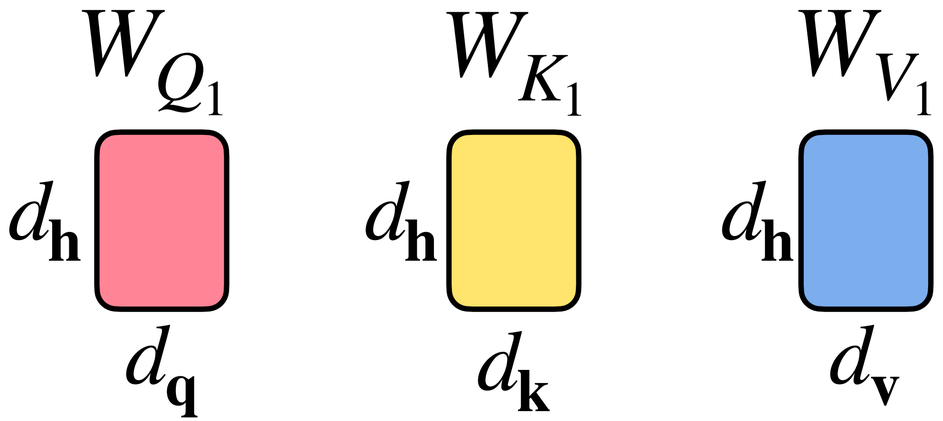
$$Z = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Multi-head Self-attention

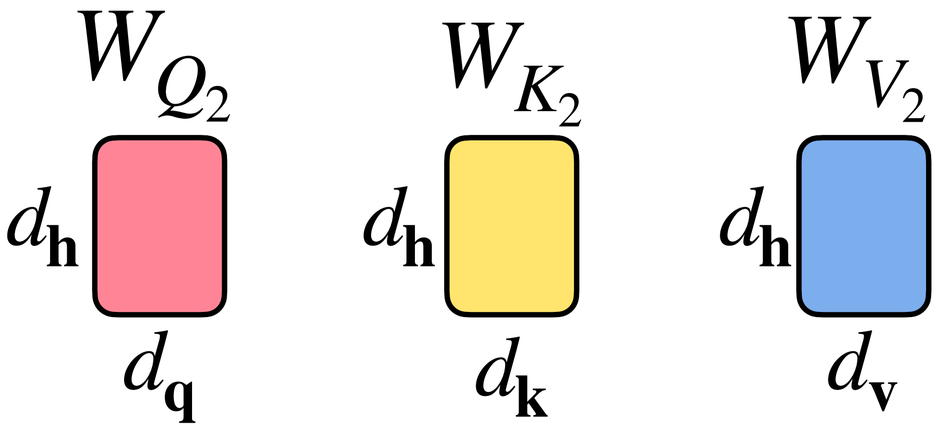
Now let's add more attention heads!



Head 1:



Head 2:

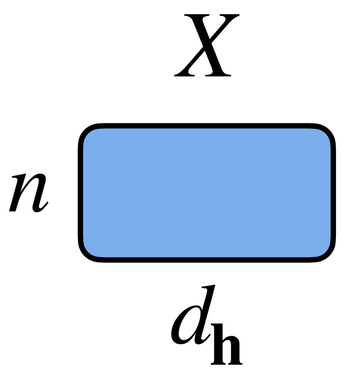


$$Z_1 = \text{Softmax}\left(\frac{Q_1 K_1^\top}{\sqrt{d_\mathbf{k}}}\right) V_1$$

$$Z_2 = \text{Softmax}\left(\frac{Q_2 K_2^\top}{\sqrt{d_\mathbf{k}}}\right) V_2$$

Multi-head Self-attention

Now let's add more attention heads!



But wait, the model expects the output of the attention block to be the size of a single Z. How do we convert all these Z_i into a Z-shaped matrix?

Concatenate then project:

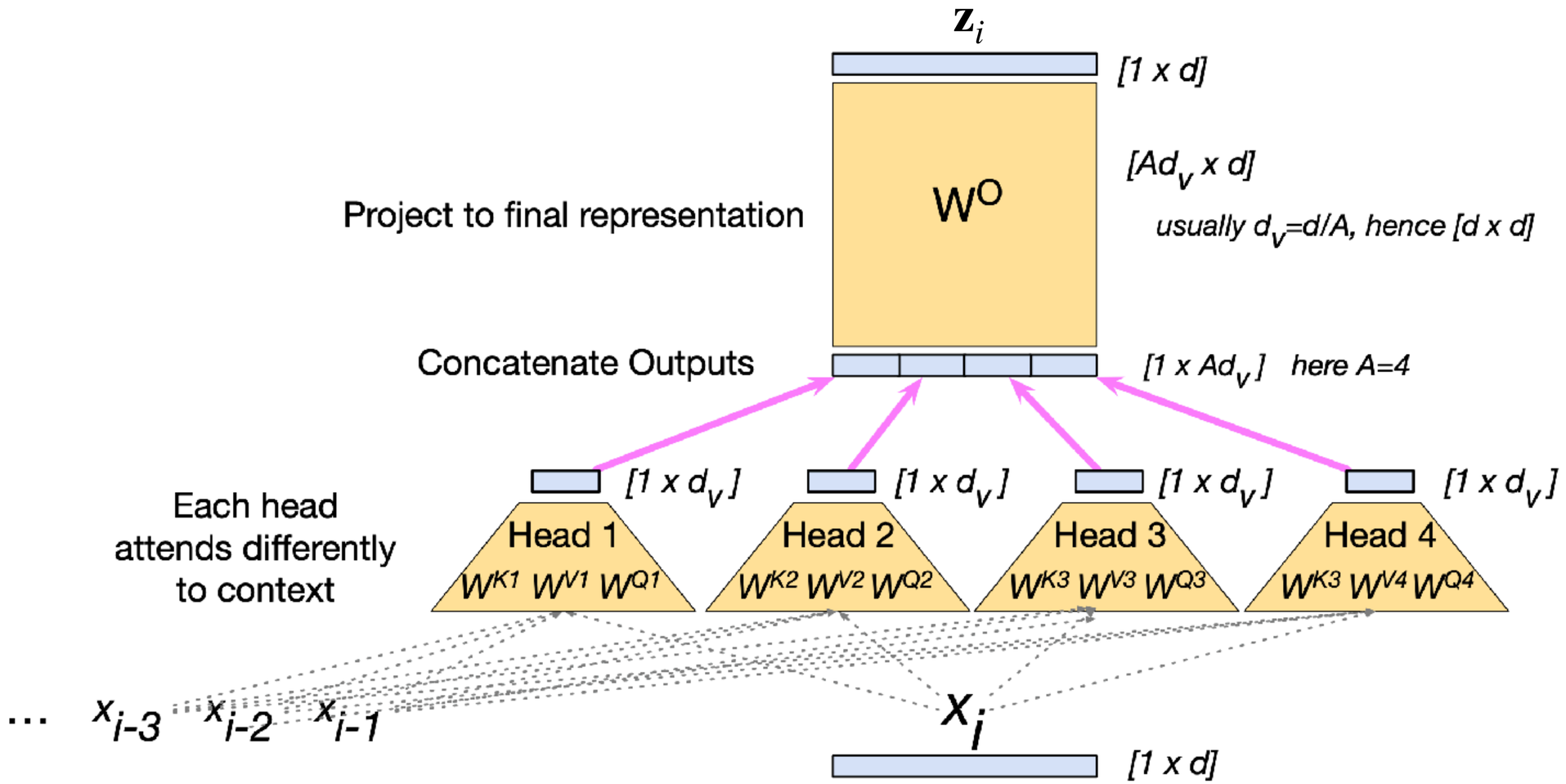
$$Z_1 = \text{Softmax}\left(\frac{Q_1 K_1^\top}{\sqrt{d_{\mathbf{k}}}}\right) V_1$$

$$Z_2 = \text{Softmax}\left(\frac{Q_2 K_2^\top}{\sqrt{d_{\mathbf{k}}}}\right) V_2$$

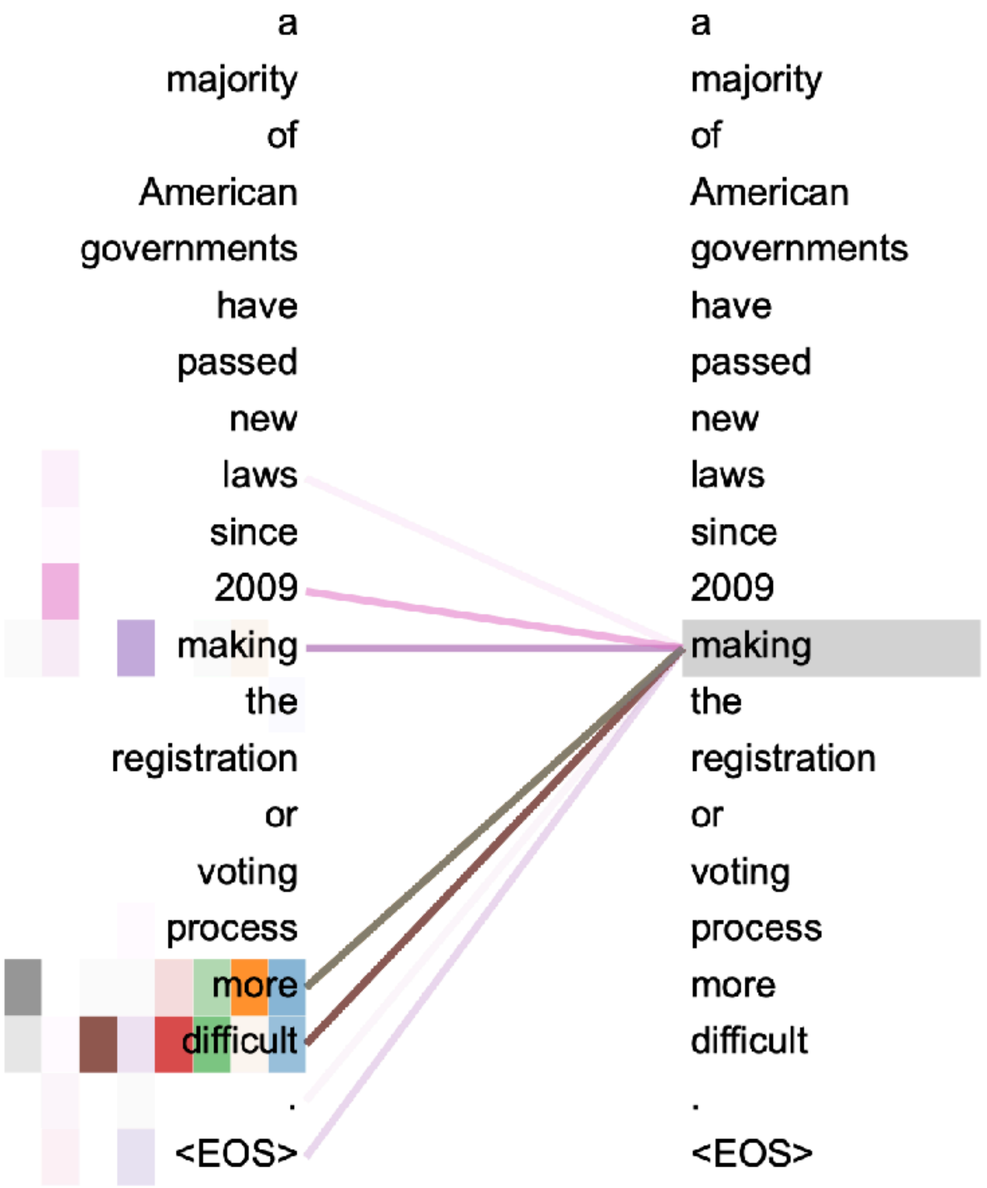
$$Z = \begin{array}{c} Z_1 \quad Z_2 \\ \square\square \end{array} W_O$$

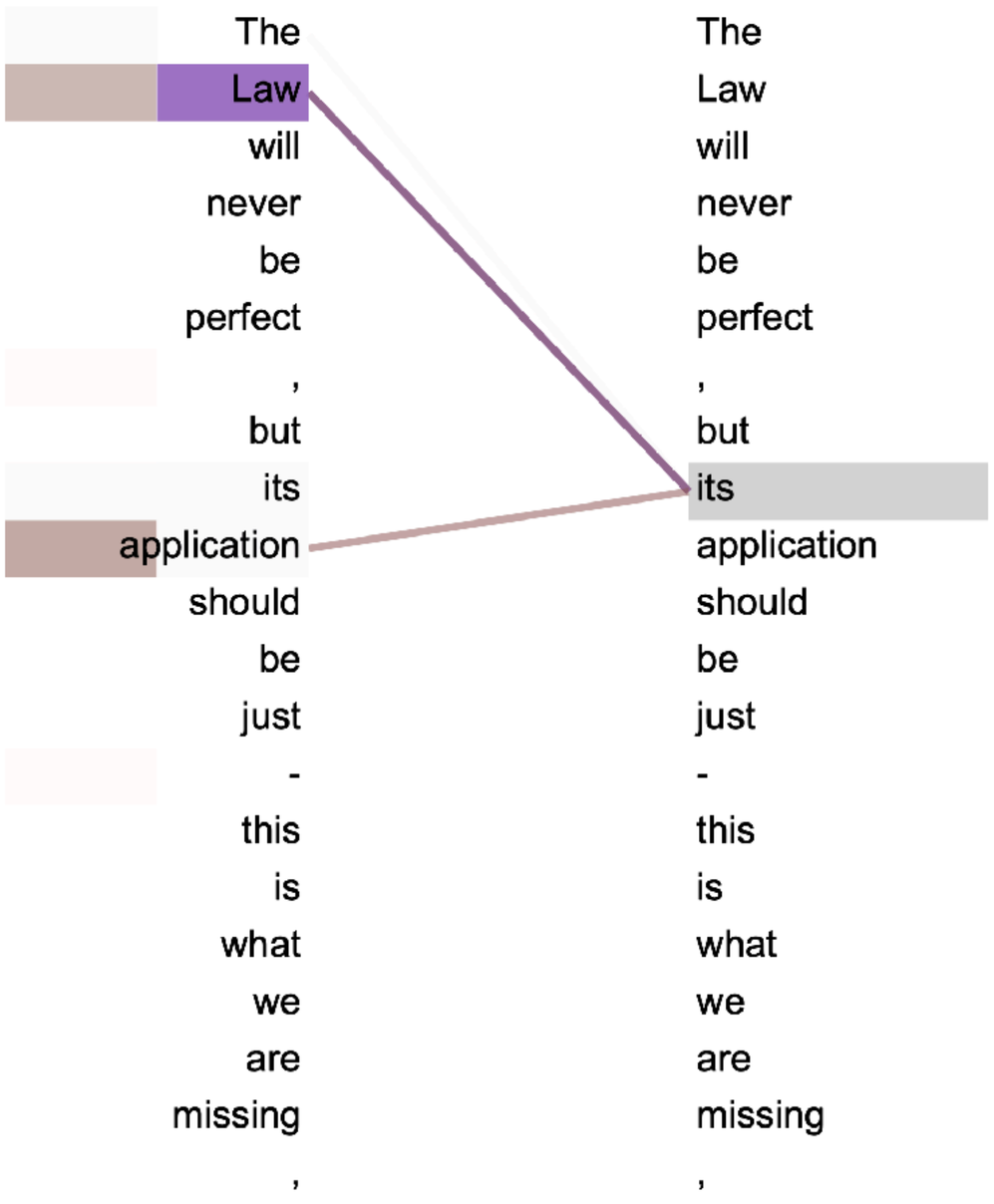
Output projection

Multi-head Self-attention



Let’s look at a couple attention heads using BERTViz:





Contextual Representations

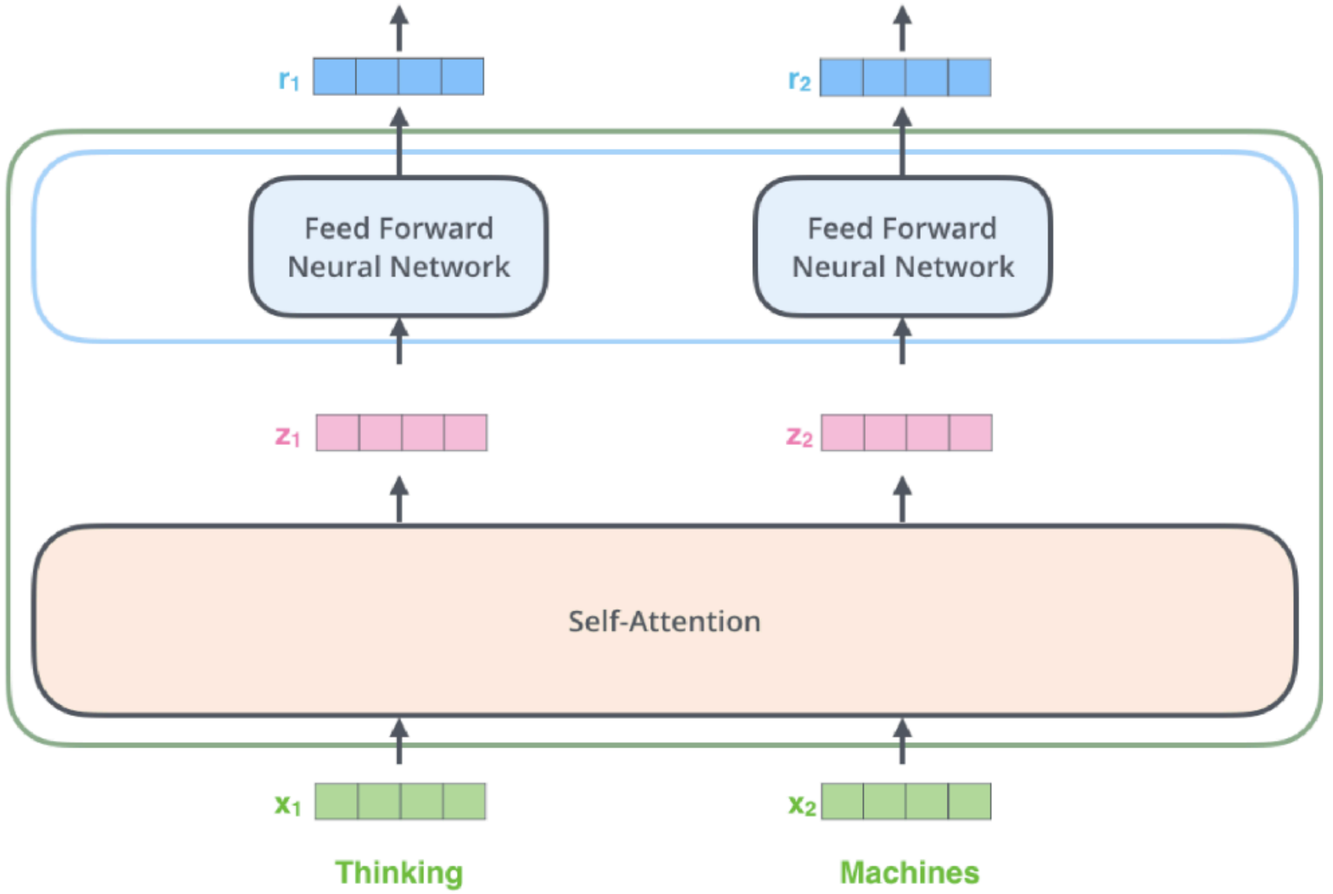
- Because they receive information from other tokens, token representations $\mathbf{r}_i$ after a Transformer block carry contextual information!

E.g., $\mathbf{r}_i$ can distinguish between:

Let's go to the **park**

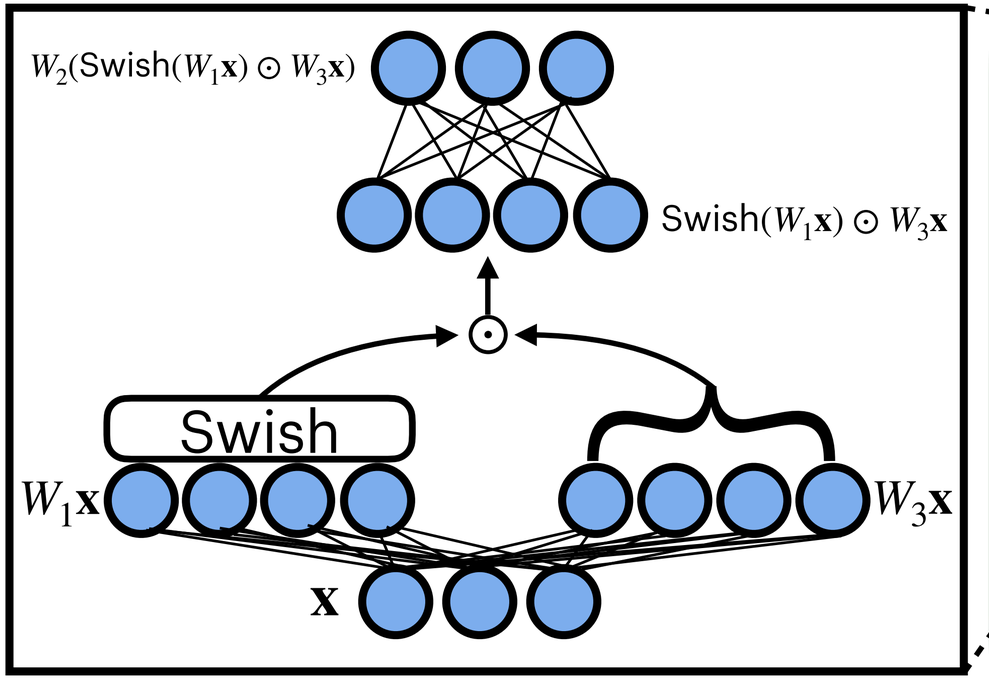
Let me **park** my car

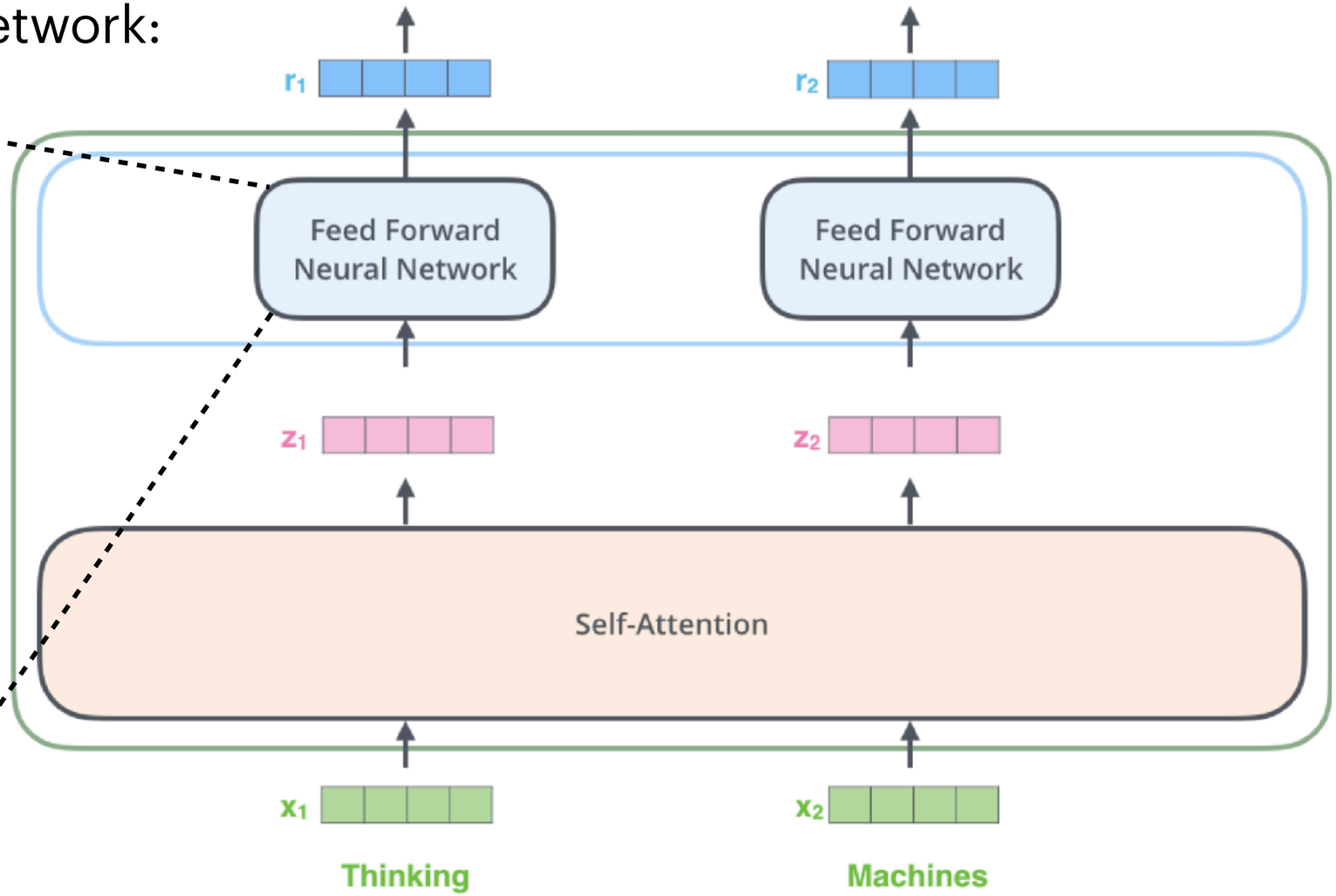
Let's **park** that for now



Attention is totally linear... so where do the non-linearities come from?

We add a SwiGLU feed-forward network:

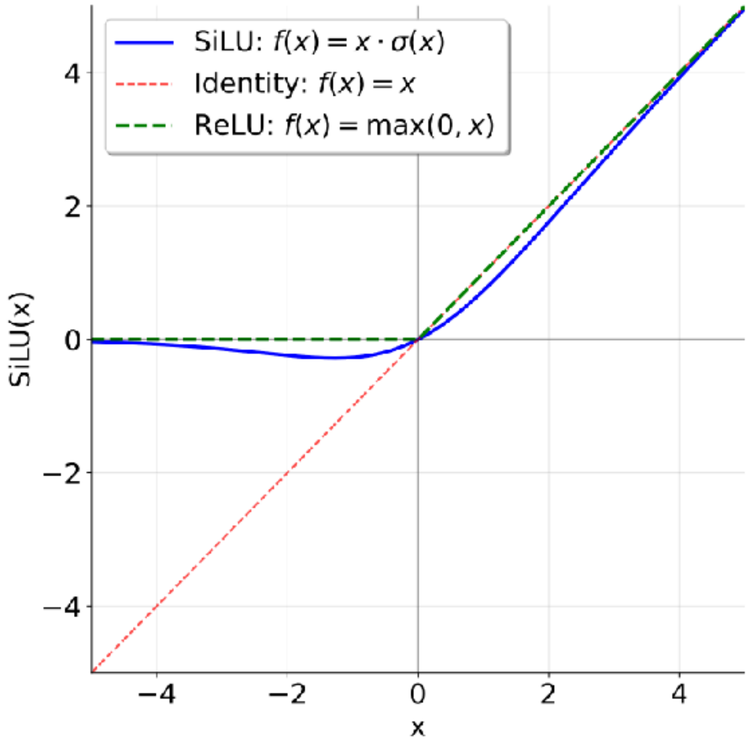




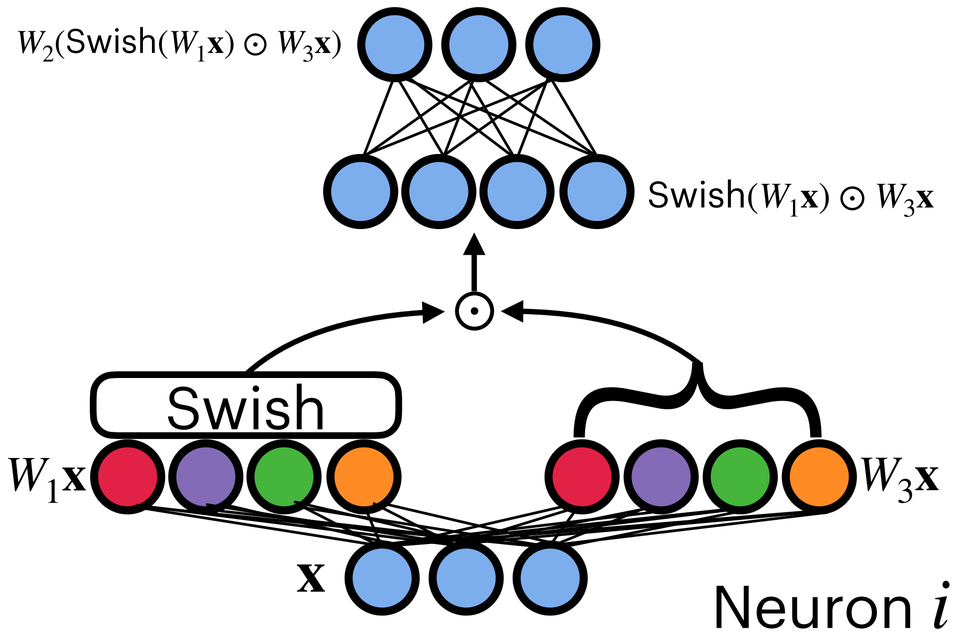
$$\text{FFN}(\mathbf{x}) = \text{SwiGLU}(x, W_1, W_2, W_3)$$
$$= W_2(\text{Swish}(W_1\mathbf{x}) \odot W_3\mathbf{x})$$

The SwiGLU

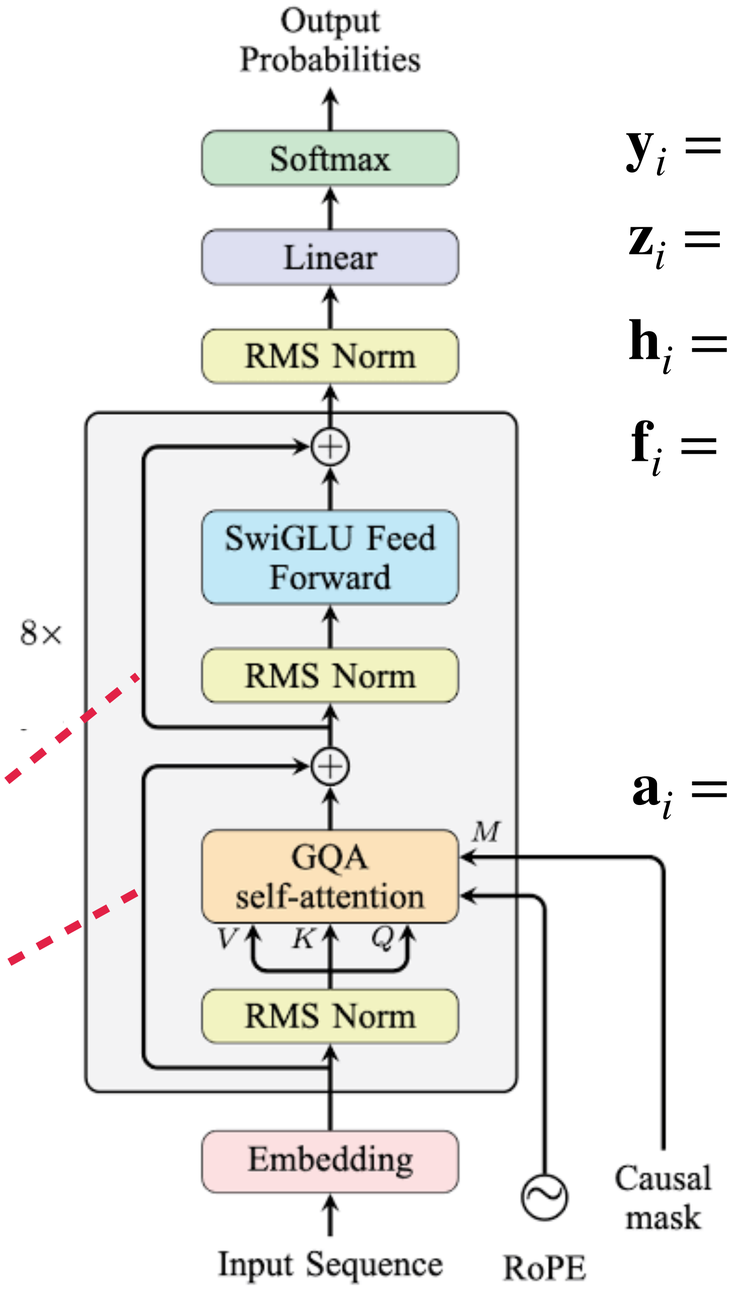
Combines the Swish function with a gated linear unit (GLU):



$$\text{Swish}(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}}$$



Neuron i of $W_3\mathbf{x}$ *gates* the Swish'd neuron i of $W_1\mathbf{x}$



$$\mathbf{y}_i = \text{softmax}(\mathbf{z}_i)$$

$$\mathbf{z}_i = W_f \mathbf{h}_i$$

$$\mathbf{h}_i = \text{RMSNorm}(\mathbf{f}_i)$$

$$\mathbf{f}_i = \text{MLP}((\text{RMSNorm}(\mathbf{a}_i)) + \mathbf{a}_i$$

$$\mathbf{a}_i = \text{MHSelfAttn}(\text{RMSNorm}(\mathbf{x}_i)) + \mathbf{x}_i$$

$$\mathbf{x}_i = \text{Embedding}(t_i)$$

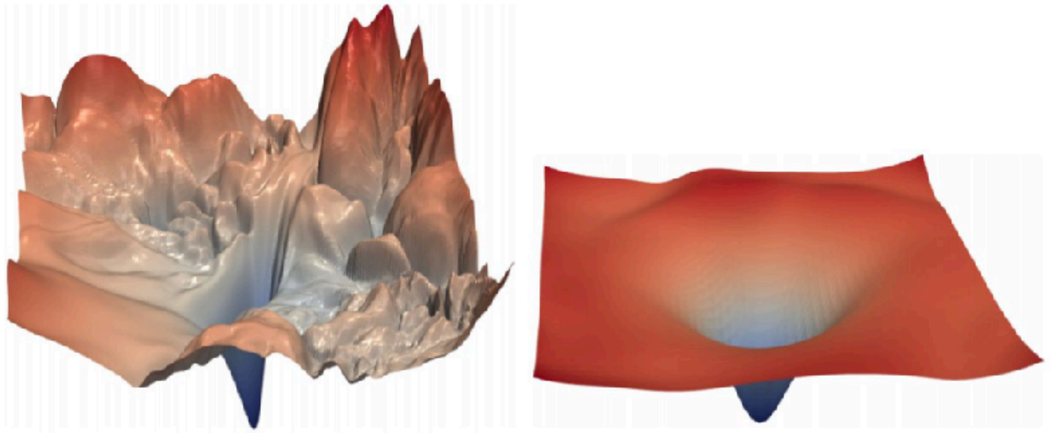
Residual connections help prevent vanishing gradients. Gives gradients more direct paths from later layers to earlier layers.

Why residual connections?

1. It helps gradients not vanish as easily.

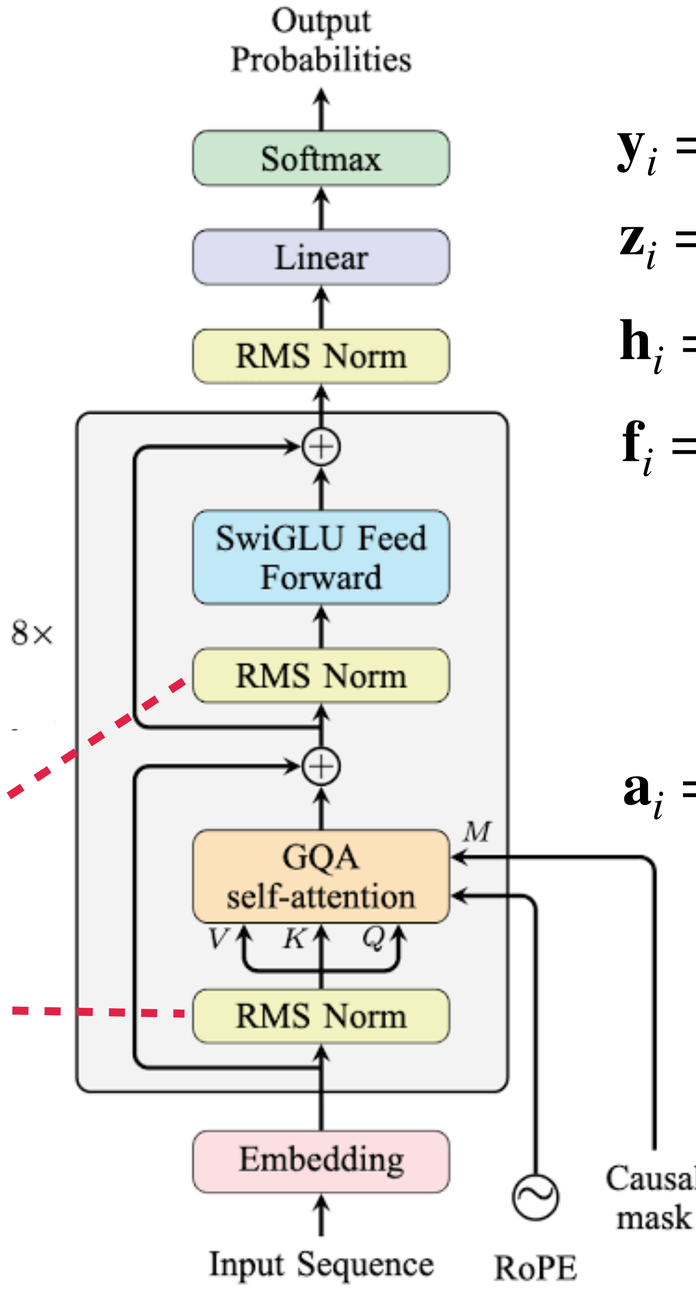
2. It makes the loss landscape much easier to optimize over:

Loss landscapes:



(no residuals) (with residuals)

[Li et al., 2018]



$$\mathbf{y}_i = \text{softmax}(\mathbf{z}_i)$$

$$\mathbf{z}_i = W_f \mathbf{h}_i$$

$$\mathbf{h}_i = \text{RMSNorm}(\mathbf{f}_i)$$

$$\mathbf{f}_i = \text{MLP}(\text{RMSNorm}(\mathbf{a}_i)) + \mathbf{a}_i$$

$$\mathbf{a}_i = \text{MHSelfAttn}(\text{RMSNorm}(\mathbf{x}_i)) + \mathbf{x}_i$$

$$\mathbf{x}_i = \text{Embedding}(t_i)$$

LayerNorm helps the model learn faster by cutting down on uninformative variance. Helps normalize the gradients.

RMSNorm

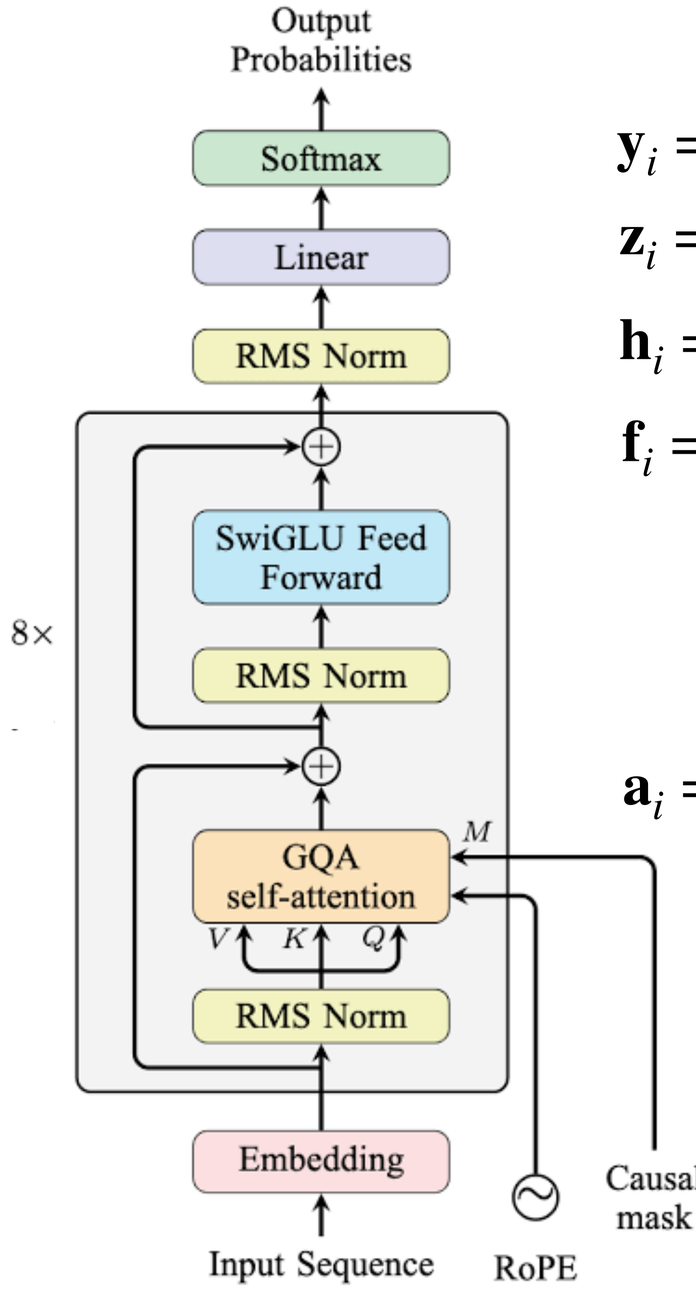
- It helps models train faster.
  - Cuts down on uninformative variance in hidden vectors within each layer.
  - Helps normalize gradients.
- The parameters g and ϵ help determine how much we should normalize:

$$\text{RMSNorm}(\mathbf{a}_i) = g_i \cdot \frac{\mathbf{a}_i}{\text{RMS}(\mathbf{a}_i)}$$

"gain"

$$\text{Where RMS}(\mathbf{a}) = \sqrt{\frac{1}{\text{length}(\mathbf{a})} \sum_{j=1}^{\text{length}(\mathbf{a})} a_j^2 + \epsilon}$$

"offset"



$$\mathbf{y}_i = \text{softmax}(\mathbf{z}_i)$$

$$\mathbf{z}_i = W_f \mathbf{h}_i$$

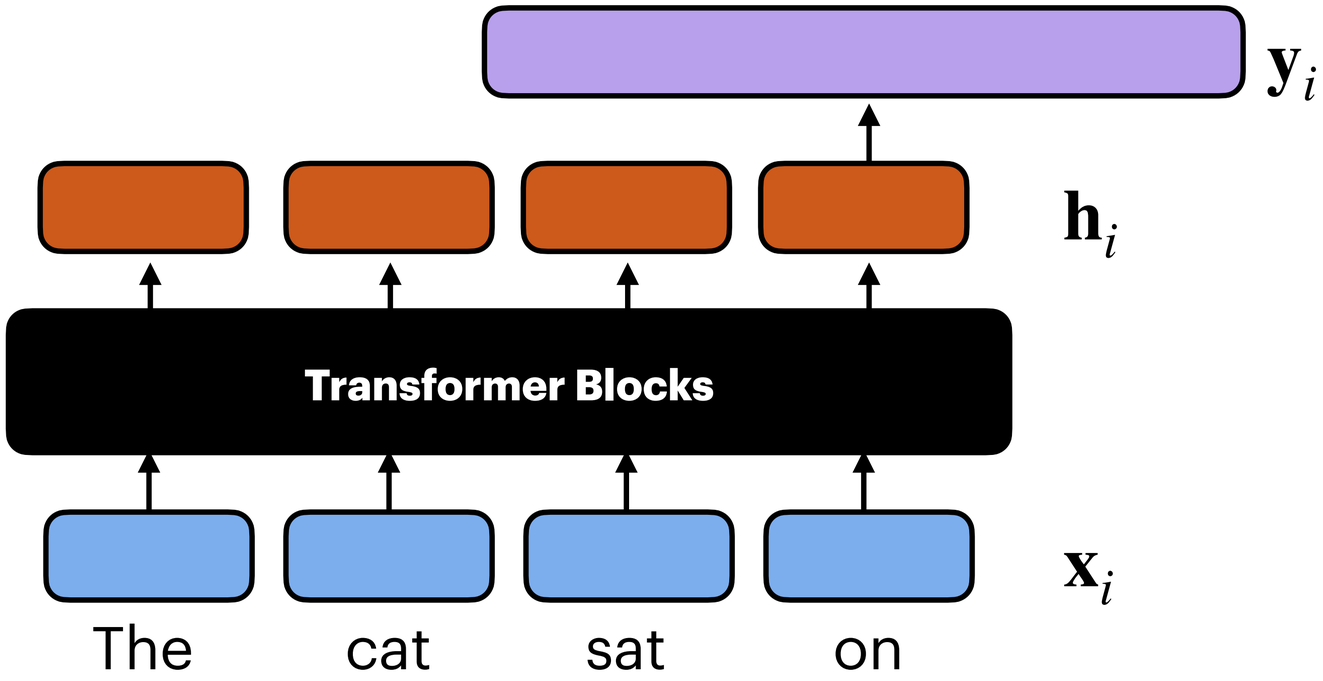
$$\mathbf{h}_i = \text{RMSNorm}(\mathbf{f}_i)$$

$$\mathbf{f}_i = \text{MLP}((\text{RMSNorm}(\mathbf{a}_i)) + \mathbf{a}_i$$

$$\mathbf{a}_i = \text{MHSelfAttn}(\text{RMSNorm}(\mathbf{x}_i)) + \mathbf{x}_i$$

$$\mathbf{x}_i = \text{Embedding}(t_i)$$

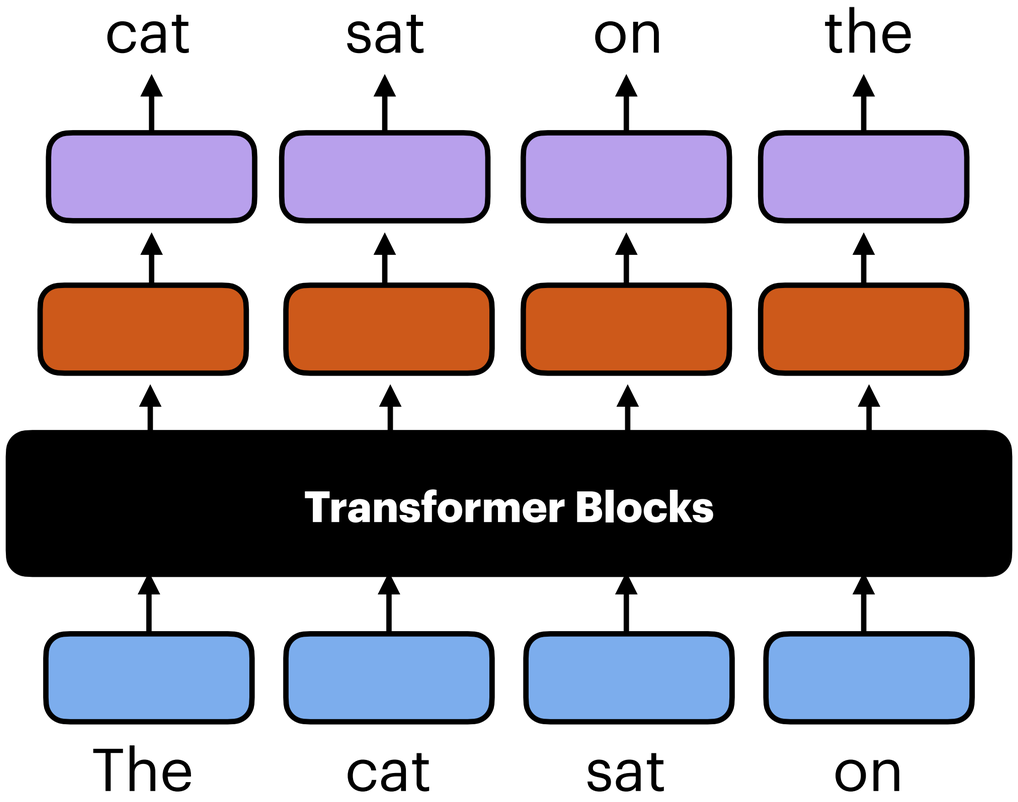
Transformer Language Modeling



$$p(x_{i+1} \mid \mathbf{x}_{1..i}) = \text{softmax}(W\mathbf{h}_i)$$

Where W is of size
(vocab size) $\times$ (hidden size)

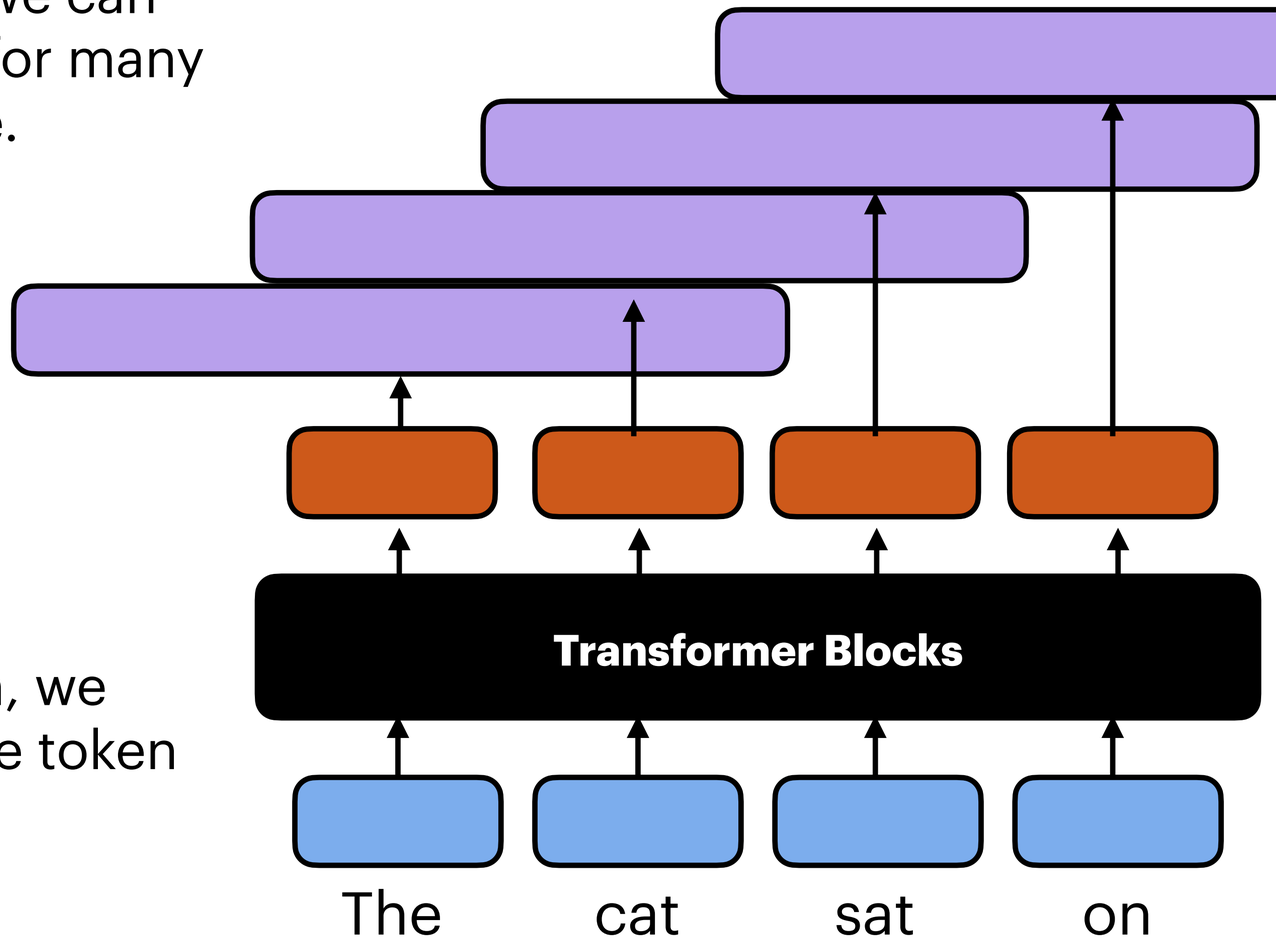
Training a Transformer LM



The input is a sequence of words, and the output is the same sequence shifted to the right by 1. This allows us to train on predictions for several positions at once.

Training a Transformer LM

During training, we can compute losses for many positions at once.



$$p(\hat{\mathbf{x}}_{i+1} \mid \mathbf{x}_{1..i})$$

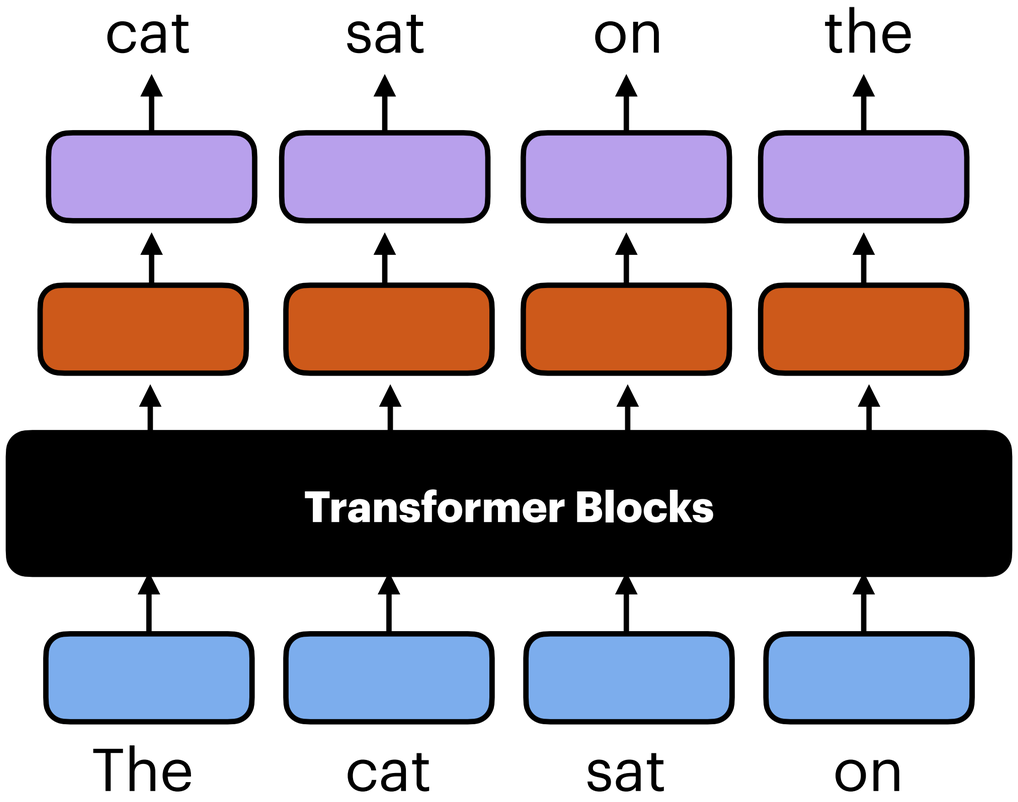
For the loss, we use **cross-entropy** (== negative log-likelihood):

$$L = -\log p(x_{i+1} \mid \mathbf{x}_{1..i})$$

The total loss for the whole sequence is simply the sum of these positionwise losses.

During generation, we must generate one token at a time.

An Issue with Full Self-attention



As-is, this model will easily get perfect accuracy. This is because it can see the correct next token during training! How can we fix this?

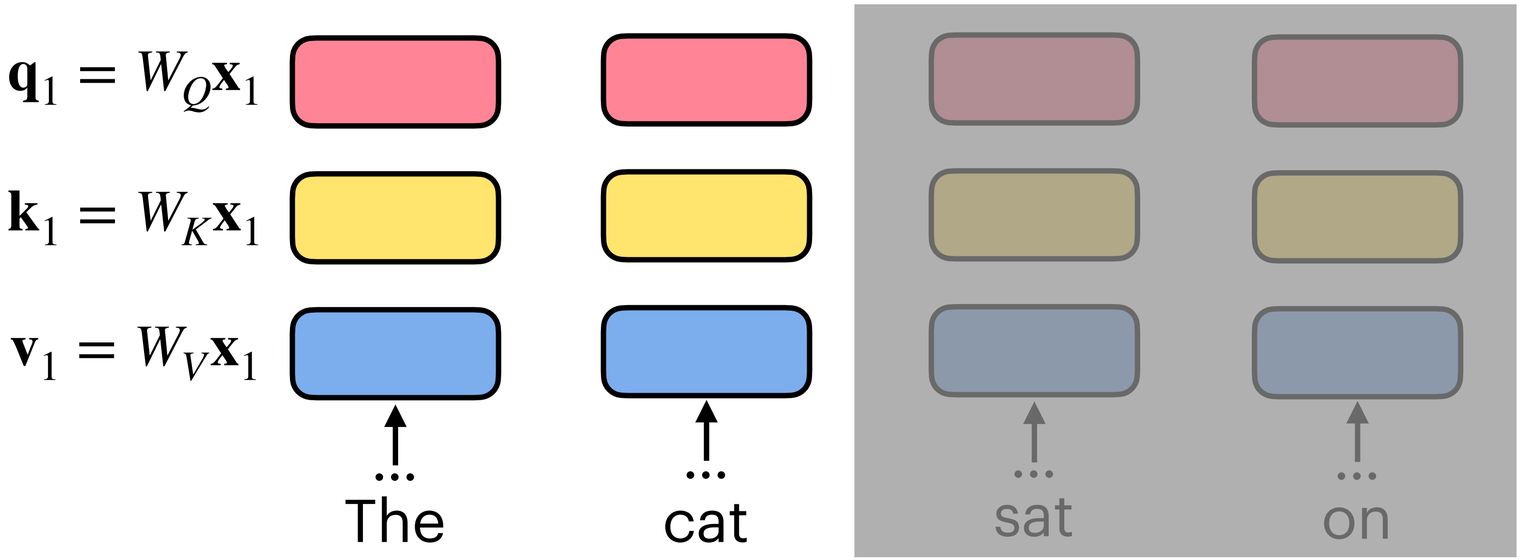
Causal Masking

Encoder: $\alpha_{ij} = \text{Softmax}\left(\frac{A_{ij}}{\sqrt{d_{\mathbf{k}}}}\right)$

Decoder: before the softmax, mask out everything above the diagonal (i.e., set those elements to $-\infty$)

At position i, mask out attention to all tokens at positions $> i$

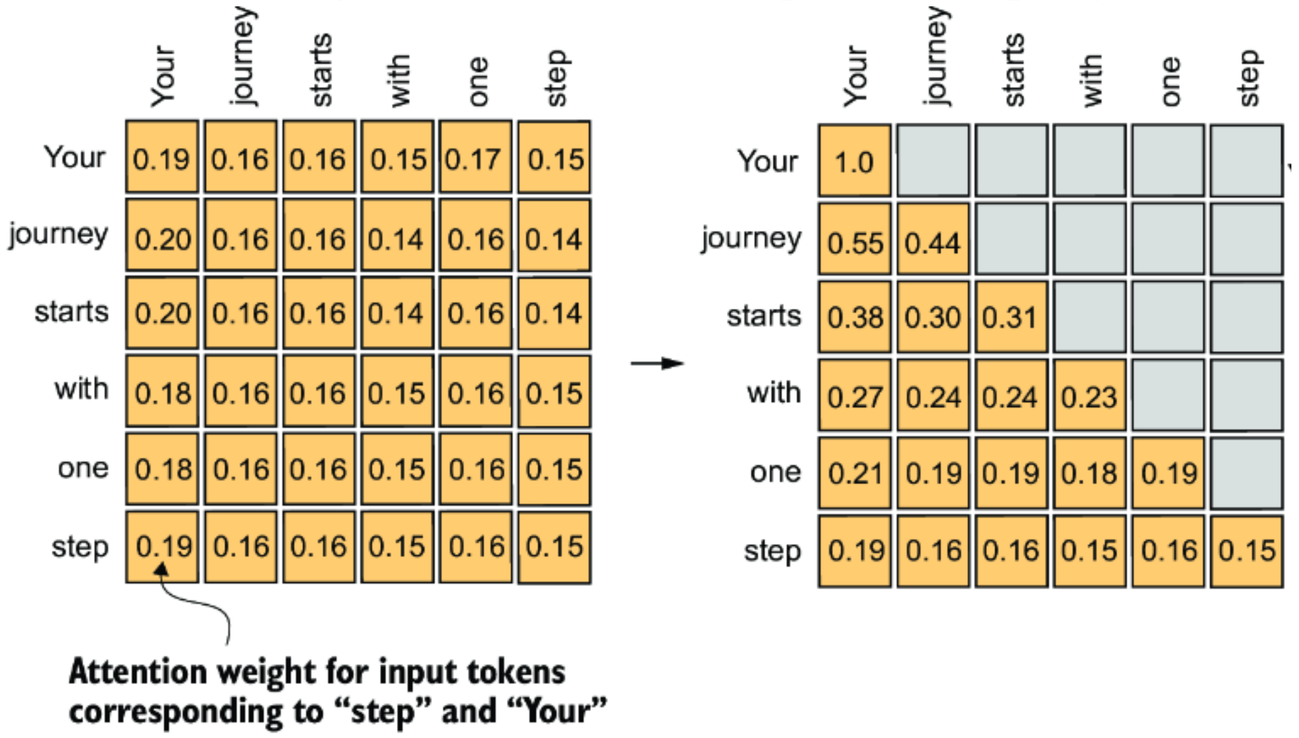
"**causal mask**"



$$\text{Softmax}\left(\begin{bmatrix} 1.1 & -\infty & -\infty & -\infty \\ -0.1 & 0.5 & -\infty & -\infty \\ 0.5 & 1.2 & 3.3 & -\infty \\ 5.1 & 2.9 & -0.3 & 4.2 \end{bmatrix}\right)$$

Why a causal mask?

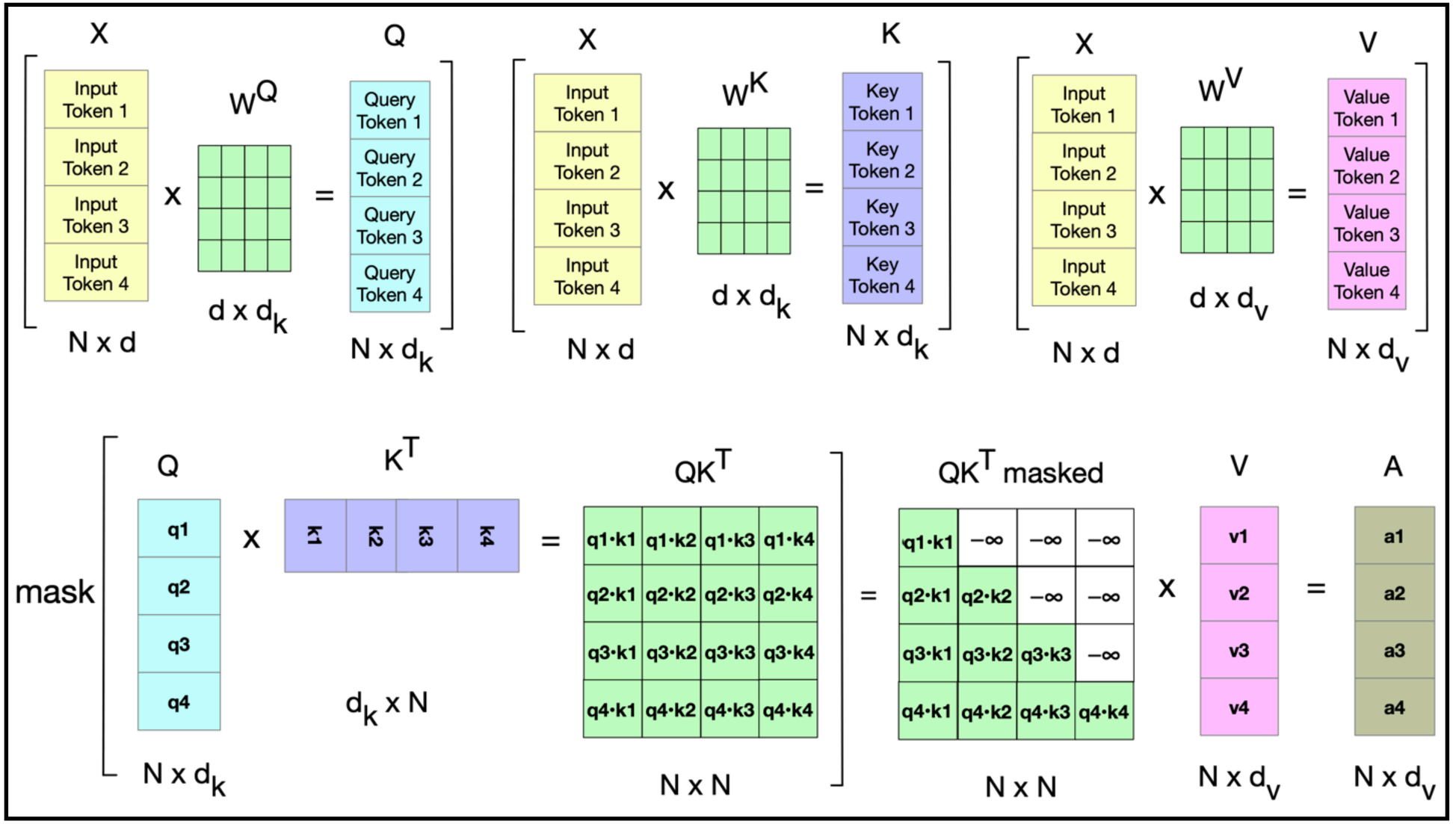
- The causal mask is what turns the Transformer into a generative language model.
  - Left-to-right information flow
- Without it, the model could just cheat during training by looking ahead at the next tokens.



Self-attention with the Causal Mask

These are all of the computations needed for one attention head.

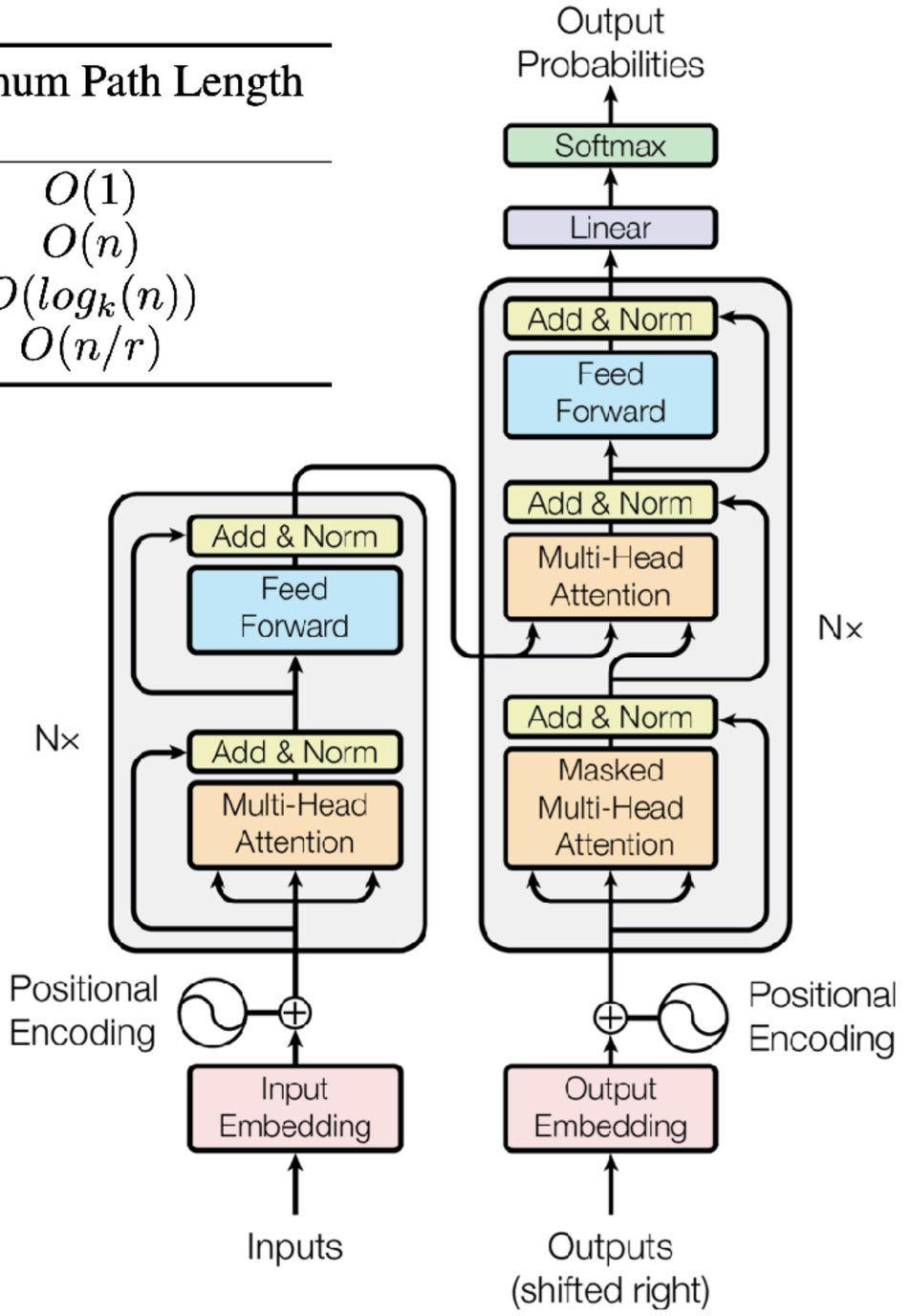
Multi-head attention is just a bunch of these performed in parallel and then recombined.



Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Transformers are technically more compute-intensive than other architectures like recurrent neural networks, but they can be massively parallelized, making Transformers much faster to train in practice.

"As [a] side benefit, self-attention could yield more interpretable models."



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	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Transformers also perform well on downstream tasks

...using an order-of-magnitude fewer FLOPs than past methods

GPT-2

The First Big Success for Transformer LMs

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	**21.8**
117M	**35.13**	45.99	**87.65**	**83.4**	**29.41**	65.85	1.16	1.17	37.50	75.20
345M	**15.60**	55.48	**92.35**	**87.1**	**22.76**	47.33	1.01	**1.06**	26.37	55.72
762M	**10.87**	**60.12**	**93.45**							
1542M	**8.63**	**63.24**	**93.30**							

Table 3. Zero-shot results on many datasets. No training
results are from (Gong et al., 2018). CBT results are from
and LAMBADA perplexity result is from (Grave et al., 20

	R-1	R-2	R-L	R-AVG
Bottom-Up Sum	**41.22**	**18.68**	**38.34**	**32.75**
Lede-3	40.38	17.66	36.62	31.55
Seq2Seq + Attn	31.33	11.81	28.83	23.99
GPT-2 `TL;DR:`	29.34	8.27	26.58	21.40
Random-3	28.78	8.63	25.52	20.98
GPT-2 no hint	21.58	4.03	19.47	15.03

Table 4. Summarization performance as measured by ROUGE F1 metrics on the CNN and Daily Mail dataset. Bottom-Up Sum is the SOTA model from (Gehrmann et al., 2018)

Impact

- With transformers, we can train neural networks that are just as good as before, but using orders-of-magnitude less compute

- Transformers significantly reduced the problem of long sequence modeling
  - (Granted, memory requirements are now quadratic w.r.t. sequence length)

- Still the dominant architecture in language modeling

- Look around us:




Some Drawbacks

- **Quadratic dependencies w.r.t. length**: we now need $O(n^2)$ compute as sequences increase in length.
  - With recurrent models, it was linear!

- The **position representations** can almost certainly be improved... more on this next time

Quadratic Dependencies

- Attention is highly parallelizable, but consider the fact that every token has pairwise interactions with every other token:

$$XQ \times K^\top X^\top = XQK^\top X^\top$$

- This means we need to perform $O(n^2 d)$ operations, where n is context length and d is the dimensionality.
- This becomes prohibitive when working with long documents.
- People have tried to remove this dependency, but methods that do so don't usually perform as well.

Some PyTorch Help

How can we implement something like $y = Wx$ without `torch.nn.Linear`?

$W \in \mathbb{R}^{in\_features \times out\_features}$

```python
class Linear(nn.Module):
    """y = Wx"""
    def __init__(self, in_features, out_features, device=None, dtype=None):
        super().__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = nn.Parameter(torch.empty(out_features, in_features, device=device, dtype=dtype))
        std = math.sqrt(2.0 / (in_features + out_features))
        nn.init.trunc_normal_(self.weight, mean=0.0, std=std, a=-3 * std, b=3 * std)

    def forward(self, x: Float[Tensor, " ... in_features"]) -> Float[Tensor, " ... out_features"]:
        return einsum(x, self.weight, "... in_features, out_features in_features -> ... out_features")
```

`__init__` creates the matrix W

`trunc_normal_` initializes the parameters
(HW1 will tell you to use this)

`forward` multiplies W with x (more on `einsum` in a later lecture)

Some PyTorch Help

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$
$$W_O$$

```python
class CausalMultiHeadSelfAttention(nn.Module):
    def __init__(
        self,
        d_model,
        num_heads,
        rope: RotaryPositionalEmbedding | None = None,
        device=None,
        dtype=None,
    ):
        super().__init__()
        assert d_model % num_heads == 0
        self.num_heads = num_heads
        self.d_head = d_model // num_heads
        self.rope = rope

        self.q_proj = Linear(d_model, d_model, device=device, dtype=dtype)
        self.k_proj = Linear(d_model, d_model, device=device, dtype=dtype)
        self.v_proj = Linear(d_model, d_model, device=device, dtype=dtype)
        self.output_proj = Linear(d_model, d_model, device=device, dtype=dtype)
```

Next Class

- Thursday: tokenization and positional embeddings
  - Byte-pair encoding (BPE)
  - Rotary position embeddings (RoPE)

- Reminder: HW1 is due **Sep. 24** at 11:59pm
  - Come to office hours with questions!

- If you haven’t yet enrolled in the class Piazza, please do that