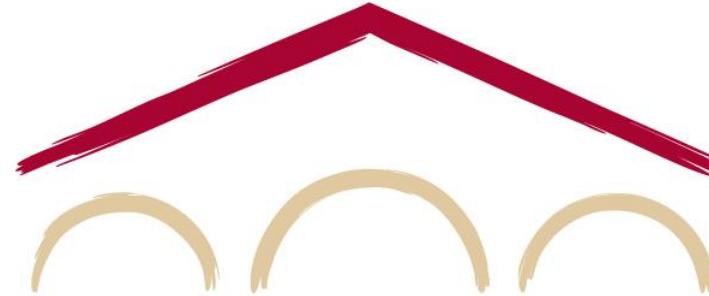


Natural Language Processing with Deep Learning

CS224N/Ling284



Anna Goldie

Lecture 9: Transformers

Announcements

- CS224n 2022 Mid-Quarter Feedback Survey
 - Your feedback is very helpful for us, so please fill it out by next Tuesday 2/8.
- There have been some issues with Azure onboarding, so we are granting the following extensions:
 - Assignment 4 is now due on Feb 8!
 - Assignment 5 is now due on Feb 17!
- Final project proposal are still due on Feb 8, so please manage your time accordingly.
- Warning: For future assignments, we cannot guarantee that we will not deduct points for not tagging properly.
- Apply for CURIS! Some NLP projects on offer:
 - <https://curis.stanford.edu/summer/>



新年快乐！

Lecture Plan

1. Impact of Transformers on NLP (and ML more broadly)
2. From Recurrence (RNNs) to Attention-Based NLP Models
3. Understanding the Transformer Model
4. Drawbacks and Variants of Transformers

Outline

1. Impact of Transformers on NLP (and ML more broadly)
2. From Recurrence (RNNs) to Attention-Based NLP Models
3. Understanding the Transformer Model
4. Drawbacks and Variants of Transformers

Transformers: Is Attention All We Need?

- Last week, we learned that attention dramatically improves the performance of recurrent neural networks.
 - Today, we will take this one step further and ask **Is Attention All We Need?**
-

Attention Is All You Need

Ashish Vaswani*

Google Brain

avaswani@google.com

Noam Shazeer*

Google Brain

noam@google.com

Niki Parmar*

Google Research

nikip@google.com

Jakob Uszkoreit*

Google Research

usz@google.com

Llion Jones*

Google Research

llion@google.com

Aidan N. Gomez* †

University of Toronto

aidan@cs.toronto.edu

Łukasz Kaiser*

Google Brain

lukaszkaiser@google.com

Illia Polosukhin* ‡

illia.polosukhin@gmail.com

Transformers: Is Attention All We Need?

- Last week, we learned that attention dramatically improves the performance of recurrent neural networks.
 - Today, we will take this one step further and ask **Is Attention All We Need?**
 - Spoiler: Not Quite!
-

Attention Is All You Need

Ashish Vaswani*

Google Brain

avaswani@google.com

Noam Shazeer*

Google Brain

noam@google.com

Niki Parmar*

Google Research

nikip@google.com

Jakob Uszkoreit*

Google Research

usz@google.com

Llion Jones*

Google Research

llion@google.com

Aidan N. Gomez* †

University of Toronto

aidan@cs.toronto.edu

Łukasz Kaiser*

Google Brain

lukaszkaiser@google.com

Illia Polosukhin* ‡

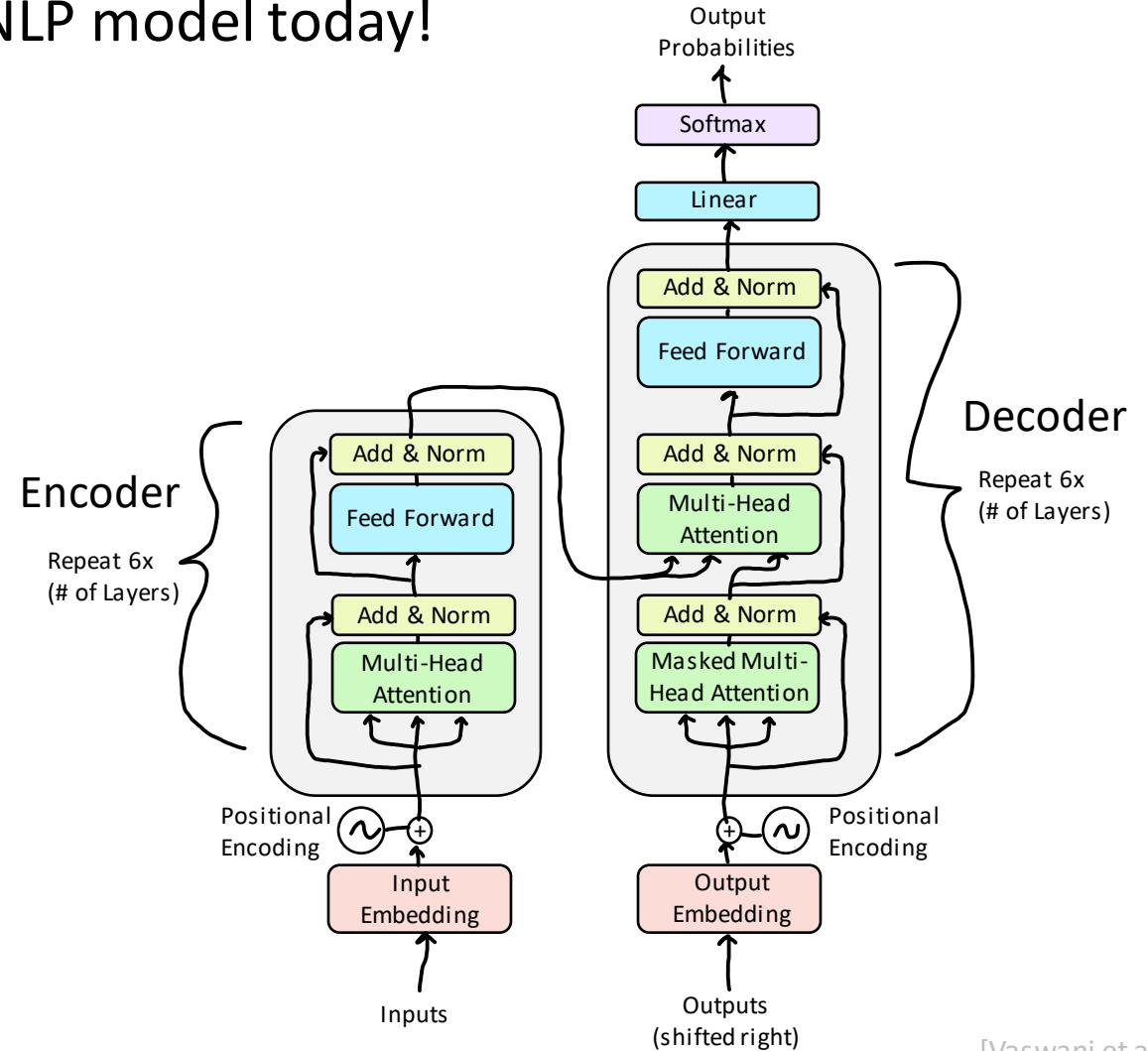
illia.polosukhin@gmail.com

Transformers Have Revolutionized the Field of NLP

- By the end of this lecture, you will deeply understand the neural architecture that underpins virtually every state-of-the-art NLP model today!



Courtesy of Paramount Pictures



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

Great Results with Transformers: Document Generation

Next, document generation!

(For perplexity, lower is better; for ROUGE-L, higher is better.)

Model	Test perplexity	ROUGE-L
<i>seq2seq-attention, L = 500</i>	5.04952	12.7
<i>Transformer-ED, L = 500</i>	2.46645	34.2
<i>Transformer-D, L = 4000</i>	2.22216	33.6
<i>Transformer-DMCA, no MoE-layer, L = 11000</i>	2.05159	36.2
<i>Transformer-DMCA, MoE-128, L = 11000</i>	1.92871	37.9
<i>Transformer-DMCA, MoE-256, L = 7500</i>	1.90325	38.8

The old standard from last week!

Transformers dominating across the board.

Preview: Great Results with (Pre-Trained) Transformers

Before too long, most Transformers results also incorporate **pretraining**, a method we'll go over on Thursday.

Transformers' parallelizability allows for efficient pretraining, and have made them the de-facto standard.

On this popular aggregate benchmark, for example:



All top models are Transformer (and pretraining)-based.

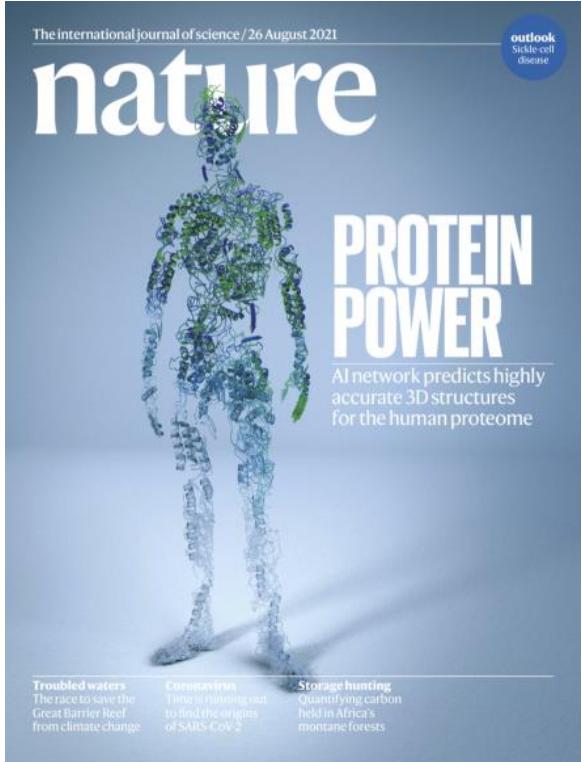
Rank	Name	Model	URL	Score
1	DeBERTa Team - Microsoft	DeBERTa / TuringNLv4	↗	90.8
2	HFL iFLYTEK	MacALBERT + DKM		90.7
3	+ Alibaba DAMO NLP	StructBERT + TAPT	↗	90.6
4	+ PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6
5	ERNIE Team - Baidu	ERNIE	↗	90.4
6	T5 Team - Google	T5	↗	90.3

More results Thursday when we discuss pretraining.

Transformers Even Show Promise Outside of NLP

Transformers Even Show Promise Outside of NLP

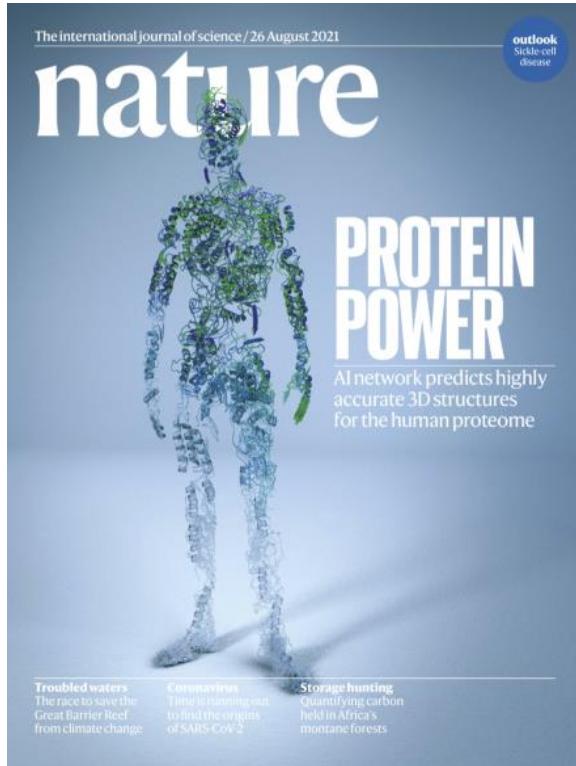
Protein Folding



[Jumper et al. 2021] aka AlphaFold2!

Transformers Even Show Promise Outside of NLP

Protein Folding



[Jumper et al. 2021] aka AlphaFold2!

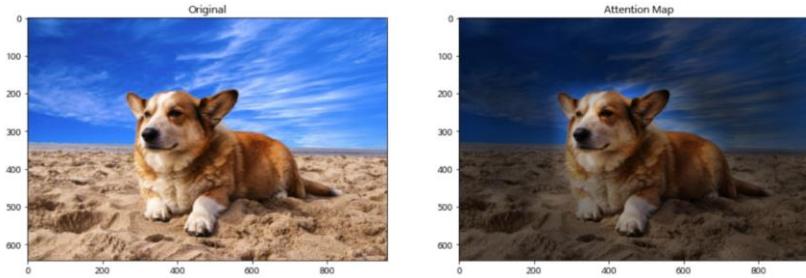


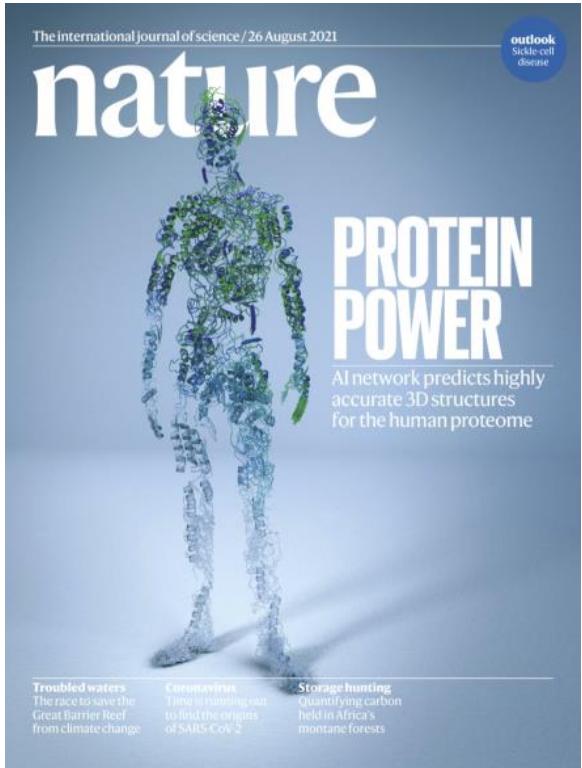
Image Classification

[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

Transformers Even Show Promise Outside of NLP

Protein Folding



[Jumper et al. 2021] aka AlphaFold2!

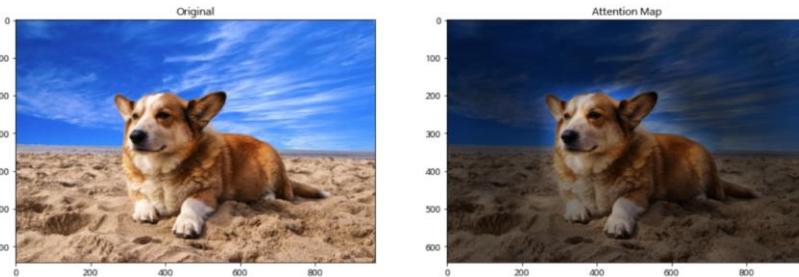
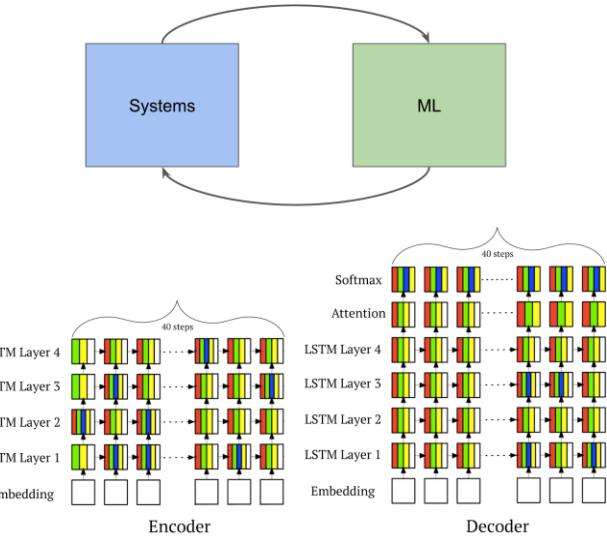


Image Classification

[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



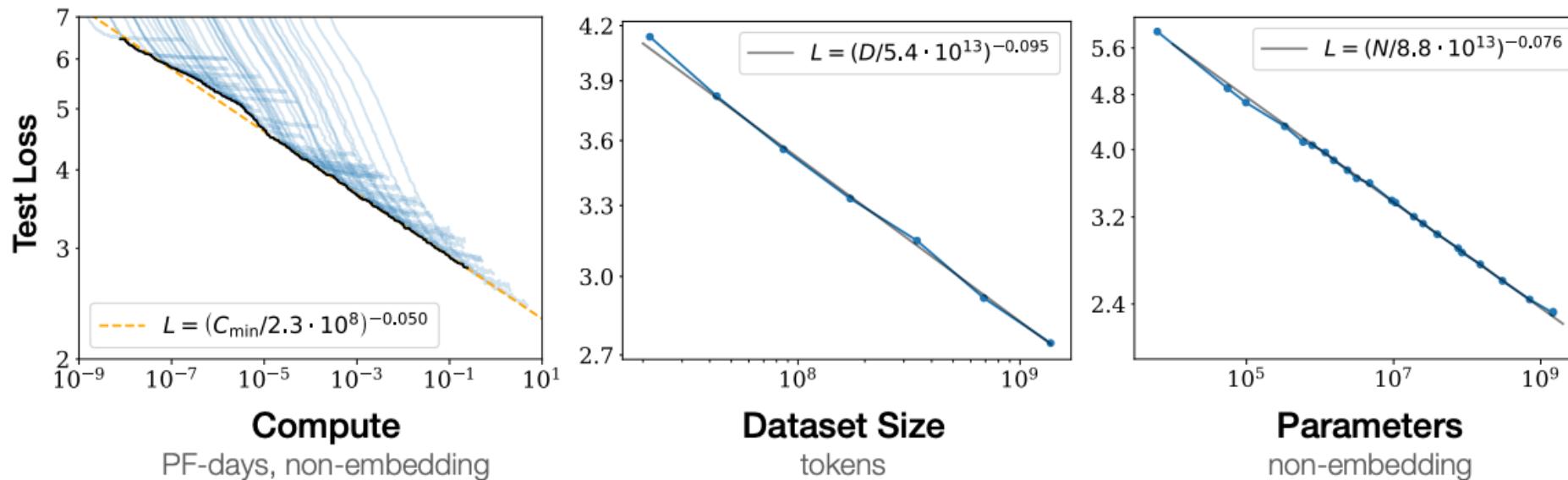
ML for Systems

[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	0.604	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
8-layer GNMT (8)	0.440	0.562	0.693	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	0.503	0.259	17.4% / 12.6%	26.7x
8-layer Transformer-XL (8)	0.350	0.46	0.604	0.425	23.9% / 16.7%	16.7x
Inception (2) b64	0.229	0.312	0.604	0.301	26.6% / 23.9%	13.5x
Inception (2) b32	0.423	0.731	0.604	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	0.604	0.354	18.6% / 11.7%	6.67x
4-stack 36-layer WaveNet (4)	0.659	0.988	0.604	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.
- 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?



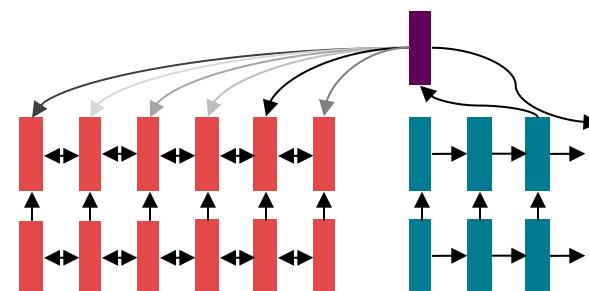
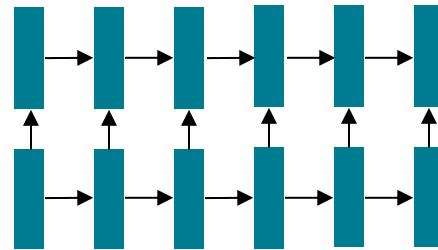
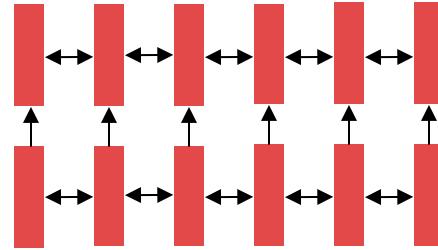
[Kaplan et al., 2020]

Outline

1. Impact of Transformers on NLP (and ML more broadly)
2. From Recurrence (RNNs) to Attention-Based NLP Models
3. Understanding the Transformer Model
4. Drawbacks and Variants of Transformers

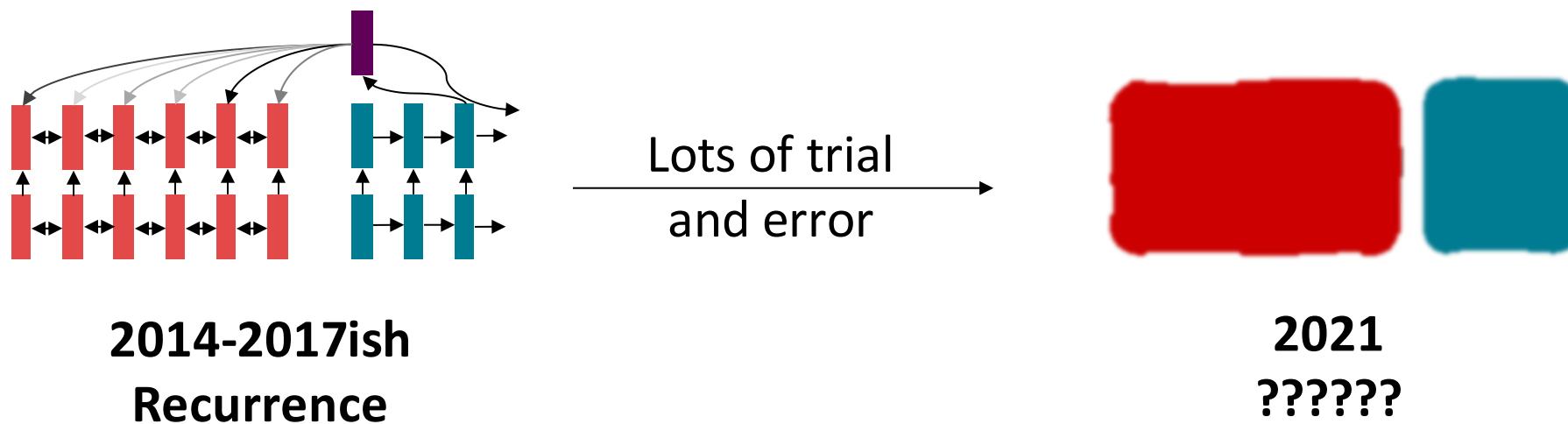
As of last week: recurrent models for (most) NLP!

- Circa 2016, the de facto strategy in NLP is to **encode** sentences with a bidirectional LSTM:
(for example, the source sentence in a translation)
- Define your output (parse, sentence, summary) as a sequence, and use an LSTM to generate it.
- Use attention to allow flexible access to memory



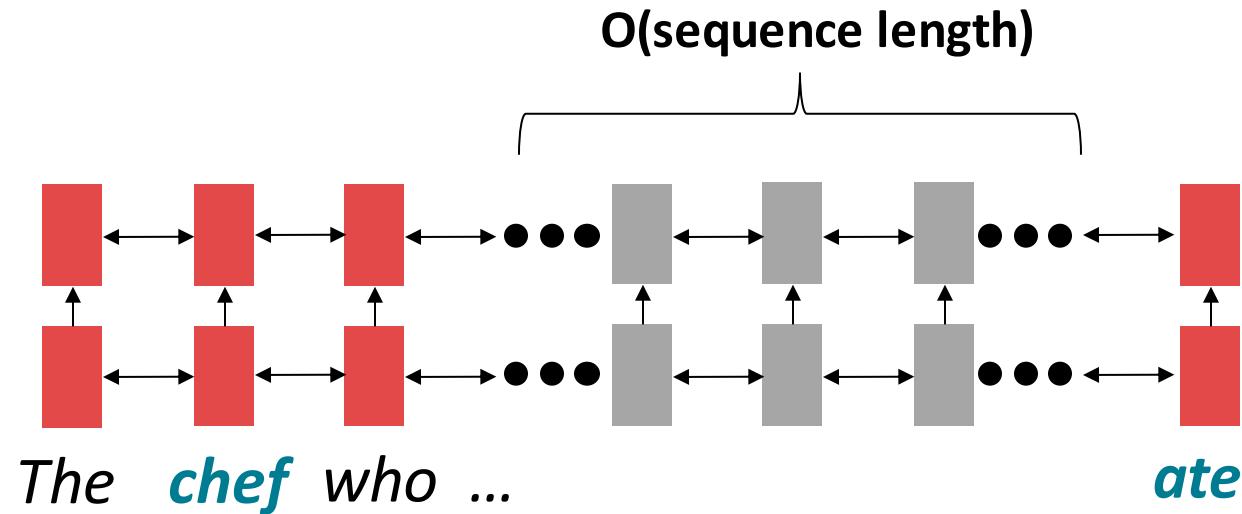
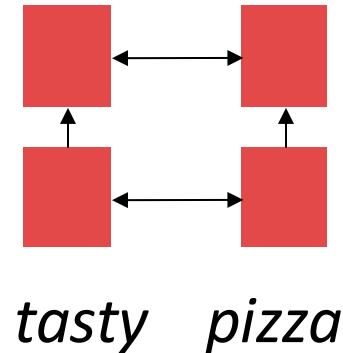
Today: Same goals, different building blocks

- Last week, we learned about sequence-to-sequence problems and encoder-decoder models.
- Today, we're **not** trying to motivate entirely new ways of looking at problems (like Machine Translation)
- Instead, we're trying to find the best **building blocks** to plug into our models and enable broad progress.



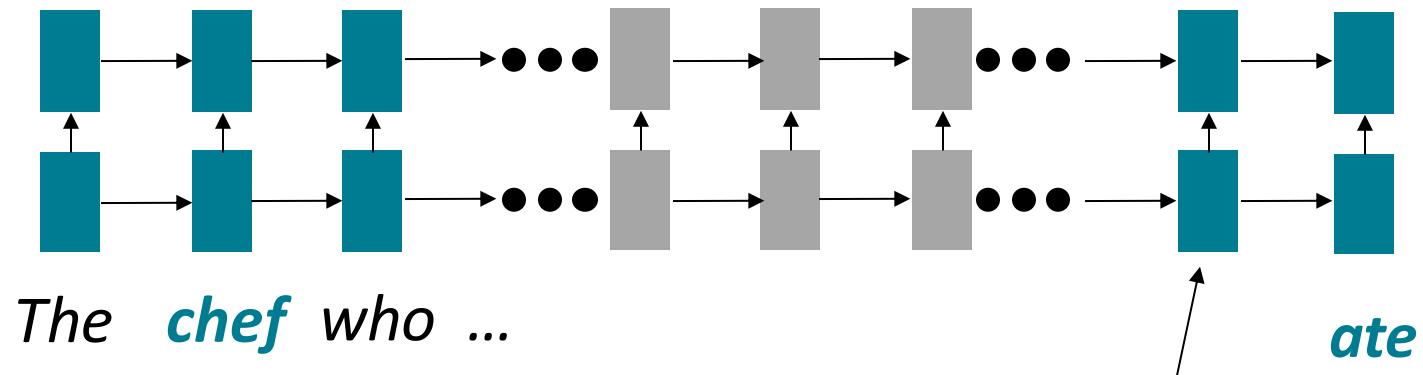
Issues with recurrent models: Linear interaction distance

- RNNs are unrolled “left-to-right”.
- It encodes linear locality: a useful heuristic!
 - Nearby words often affect each other’s meanings
- **Problem:** RNNs take **O(sequence length)** steps for distant word pairs to interact.



Issues with recurrent models: Linear interaction distance

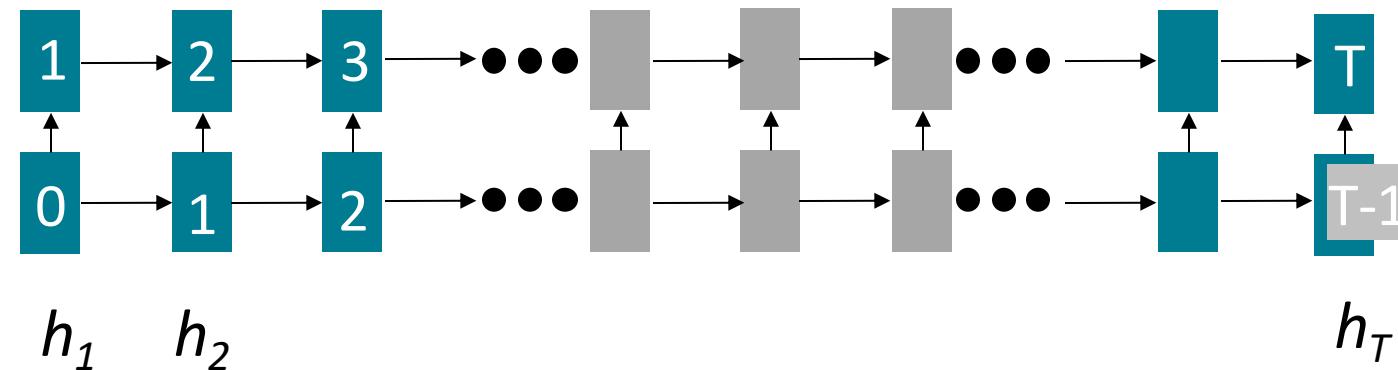
- **O(sequence length)** steps for distant word pairs to interact means:
 - Hard to learn long-distance dependencies (because gradient problems!)
 - Linear order of words is “baked in”; we already know sequential structure doesn't tell the whole story...



Info of *chef* has gone through
 $O(\text{sequence length})$ many layers!

Issues with recurrent models: Lack of parallelizability

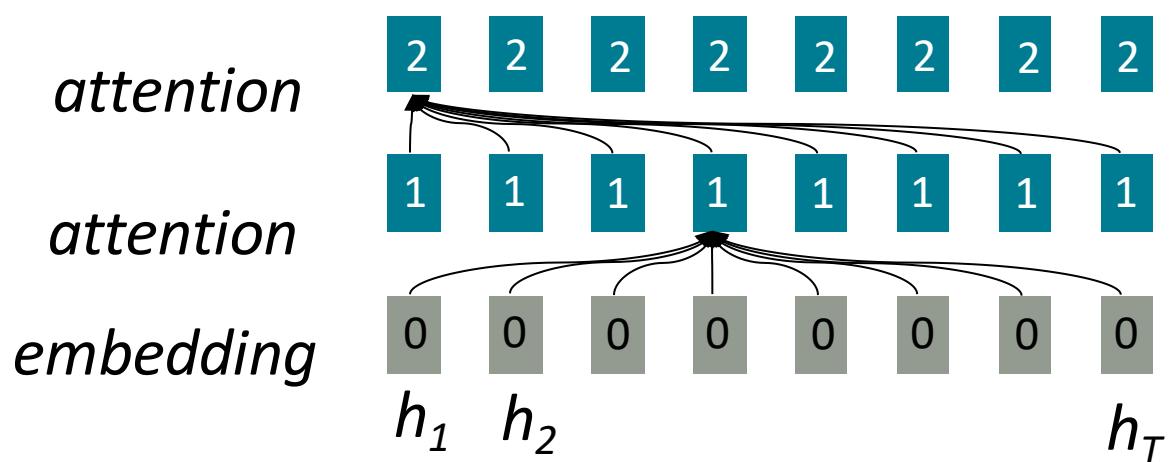
- Forward and backward passes have **O(seq length)** unparallelizable operations
 - GPUs (and TPUs) can perform many independent computations at once!
 - But future RNN hidden states can't be computed in full before past RNN hidden states have been computed
 - Inhibits training on very large datasets!
 - Particularly problematic as sequence length increases, as we can no longer batch many examples together due to memory limitations



Numbers indicate min # of steps before a state can be computed

If not recurrence, then what? How about (self) attention?

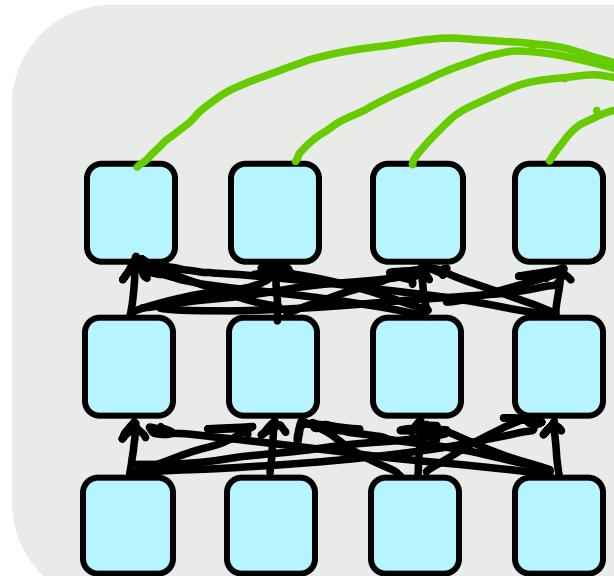
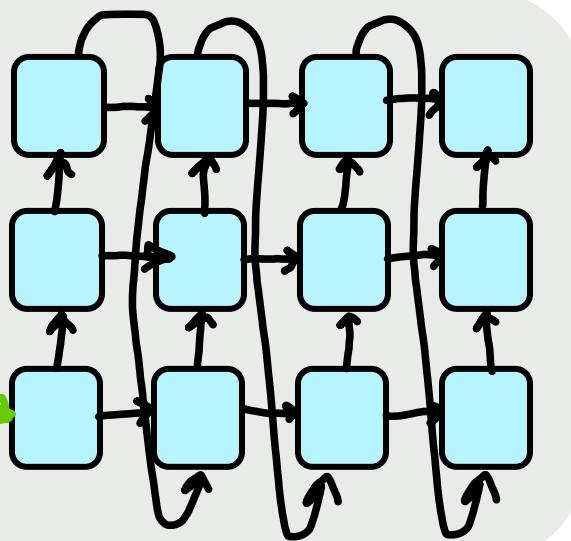
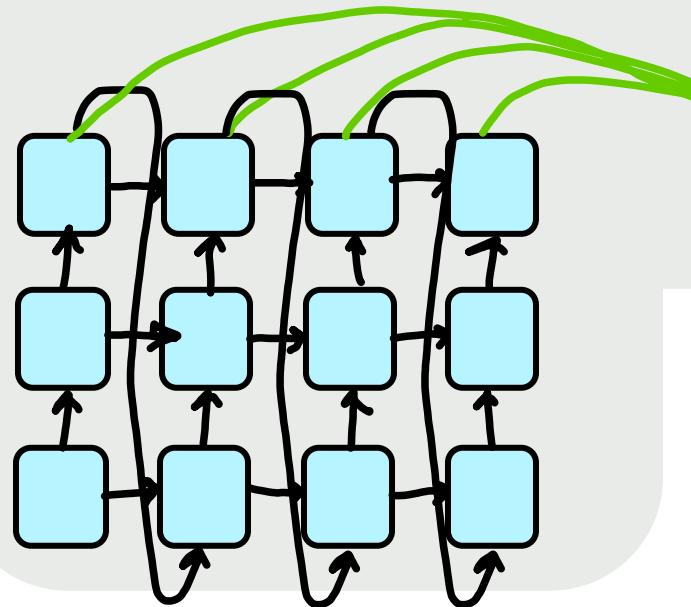
- To recap, **attention** treats each word's representation as a **query** to access and incorporate information from a **set of values**.
 - Last week, we saw attention from the **decoder** to the **encoder**;
 - Self-attention** is **encoder-encoder** (or **decoder-decoder**) attention where each word attends to each other word **within the input (or output)**.



All words attend to all words in previous layer; most arrows here are omitted

Computational Dependencies for Recurrence vs. Attention

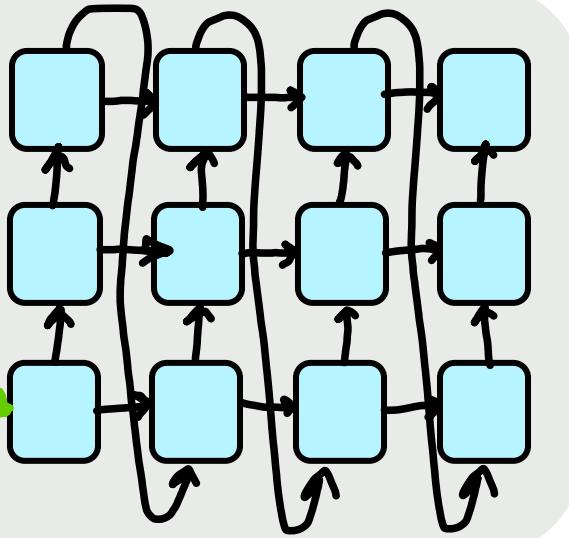
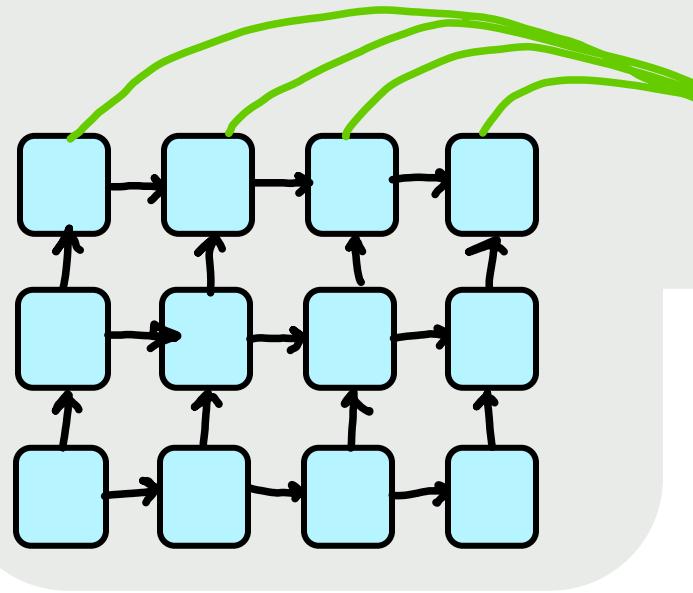
RNN-Based Encoder-Decoder Model with Attention



Transformer-Based Encoder-Decoder Model

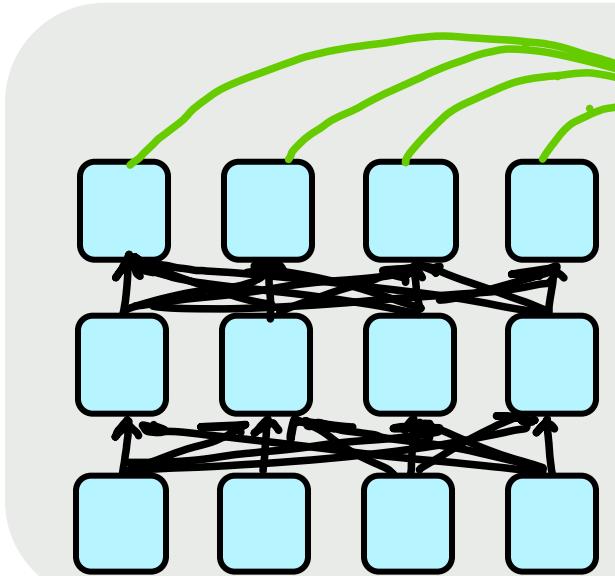
Computational Dependencies for Recurrence vs. Attention

RNN-Based Encoder-Decoder Model with Attention

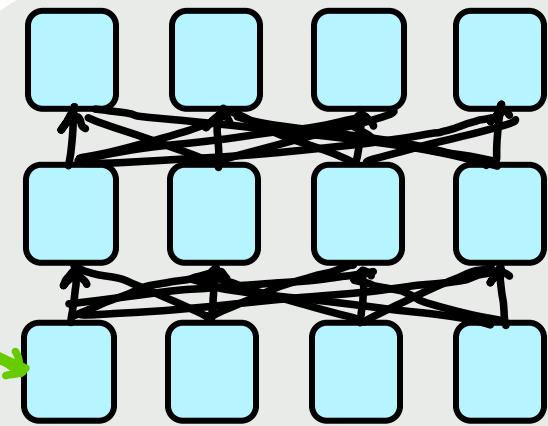


Transformer Advantages:

- Number of unparallelizable operations does not increase with sequence length.
- Each "word" interacts with each other, so maximum interaction distance: $O(1)$.



Transformer-Based Encoder-Decoder Model



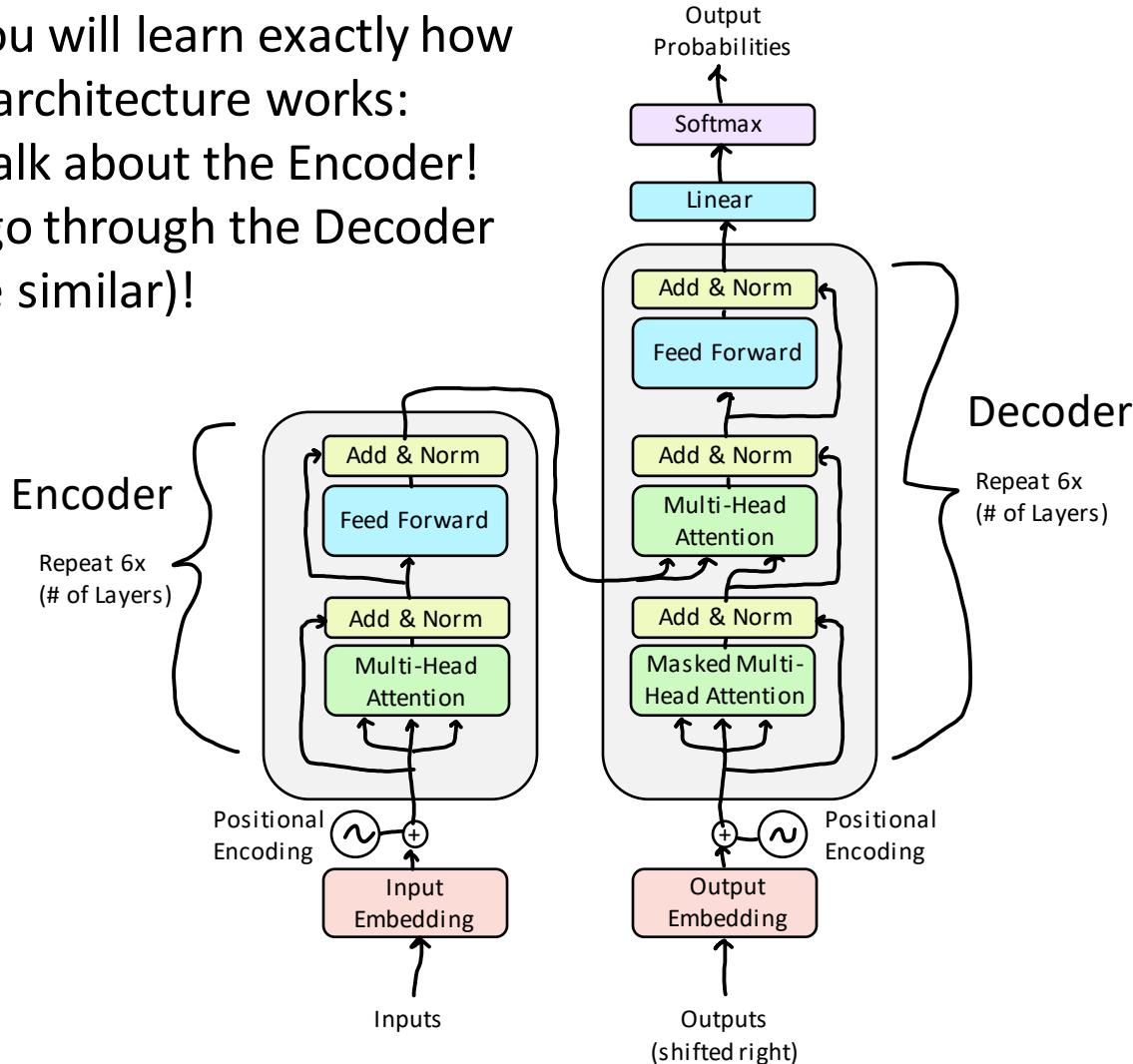
Outline

1. Impact of Transformers on NLP (and ML more broadly)
2. From Recurrence (RNNs) to Attention-Based NLP Models
3. **Understanding the Transformer Model**
4. Drawbacks and Variants of Transformers

The Transformer Encoder-Decoder [Vaswani et al., 2017]

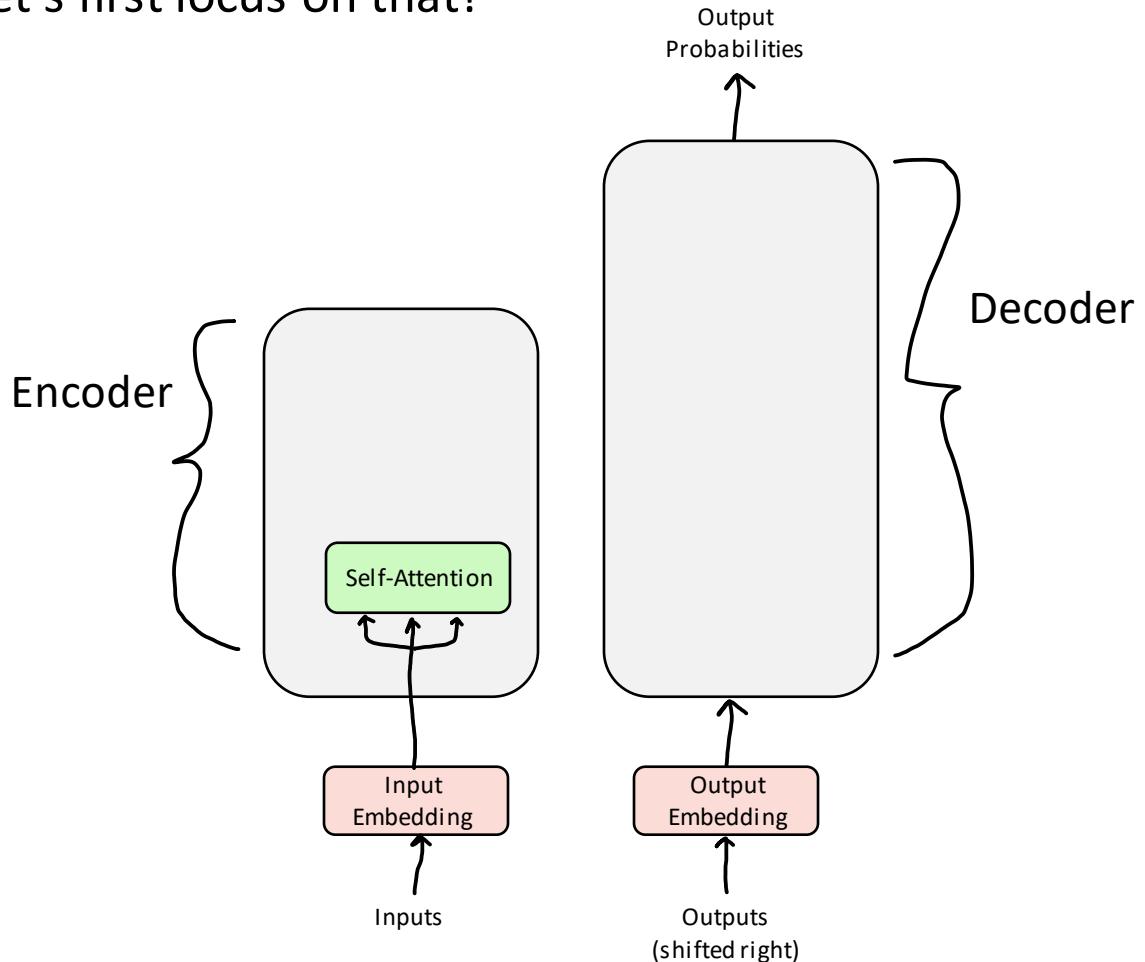
In this section, you will learn exactly how the Transformer architecture works:

- First, we will talk about the Encoder!
- Next, we will go through the Decoder (which is quite similar)!



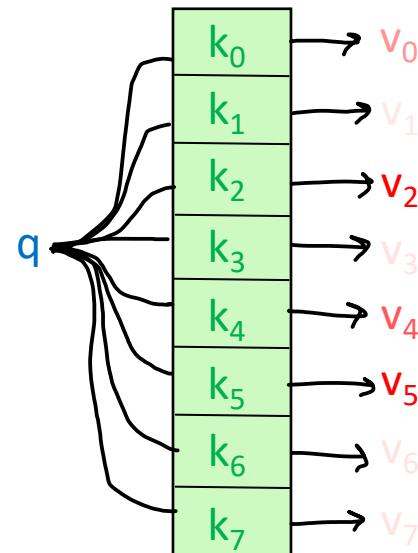
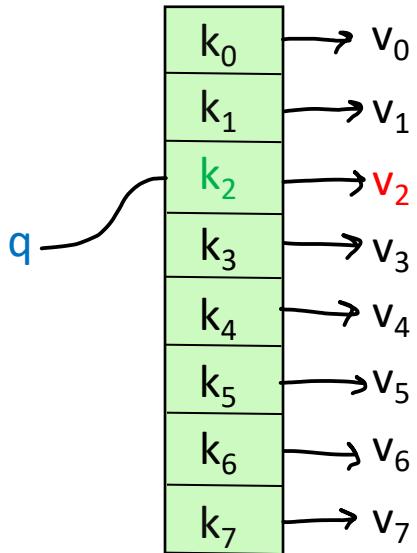
Encoder: Self-Attention

Self-Attention is the core building block of Transformer, so let's first focus on that!



Intuition for Attention Mechanism

- Let's think of attention as a "fuzzy" or approximate hashtable:
 - To look up a **value**, we compare a **query** against **keys** in a table.
 - In a hashtable (shown on the bottom left):
 - Each **query** (hash) maps to exactly one **key-value** pair.
 - In (self-)attention (shown on the bottom right):
 - Each **query** matches each **key** to varying degrees.
 - We return a sum of **values** weighted by the **query-key** match.



Recipe for Self-Attention in the Transformer Encoder

- Step 1: For each word x_i , calculate its **query**, **key**, and **value**.

$$q_i = W^Q x_i \quad k_i = W^K x_i \quad v_i = W^V x_i$$

- Step 2: Calculate attention score between **query** and **keys**.

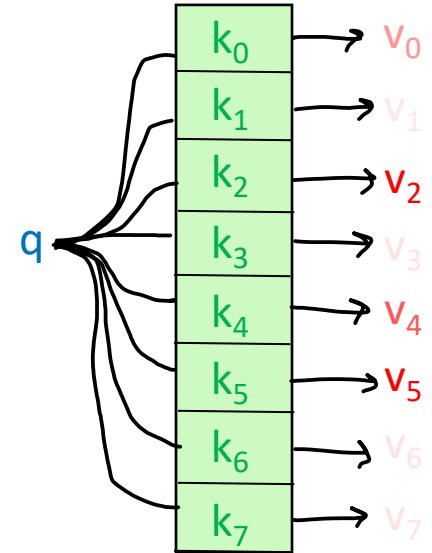
$$e_{ij} = q_i \cdot k_j$$

- Step 3: Take the softmax to normalize attention scores.

$$\alpha_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_k \exp(e_{ik})}$$

- Step 4: Take a weighted sum of **values**.

$$\text{Output}_i = \sum_j \alpha_{ij} v_j$$



Recipe for (Vectorized) Self-Attention in the Transformer Encoder

- Step 1: With embeddings stacked in X , calculate **queries**, **keys**, and **values**.

$$Q = XW^Q \quad K = XW^K \quad V = XW^V$$

- Step 2: Calculate attention scores between **query** and **keys**.

$$E = QK^T$$

- Step 3: Take the softmax to normalize attention scores.

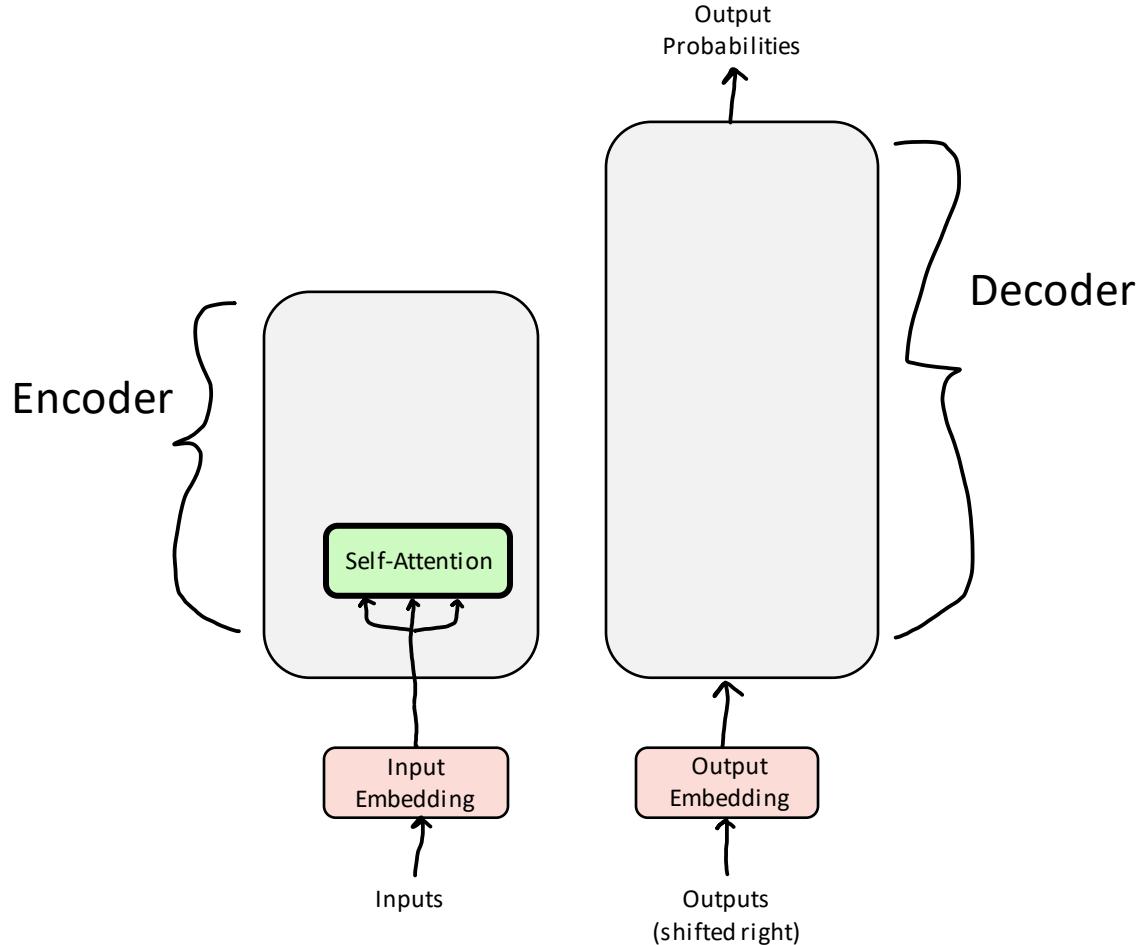
$$A = \text{softmax}(E)$$

- Step 4: Take a weighted sum of **values**.

$$\text{Output} = AV$$

$$\text{Output} = \text{softmax}(QK^T)V$$

What We Have So Far: (Encoder) Self-Attention!

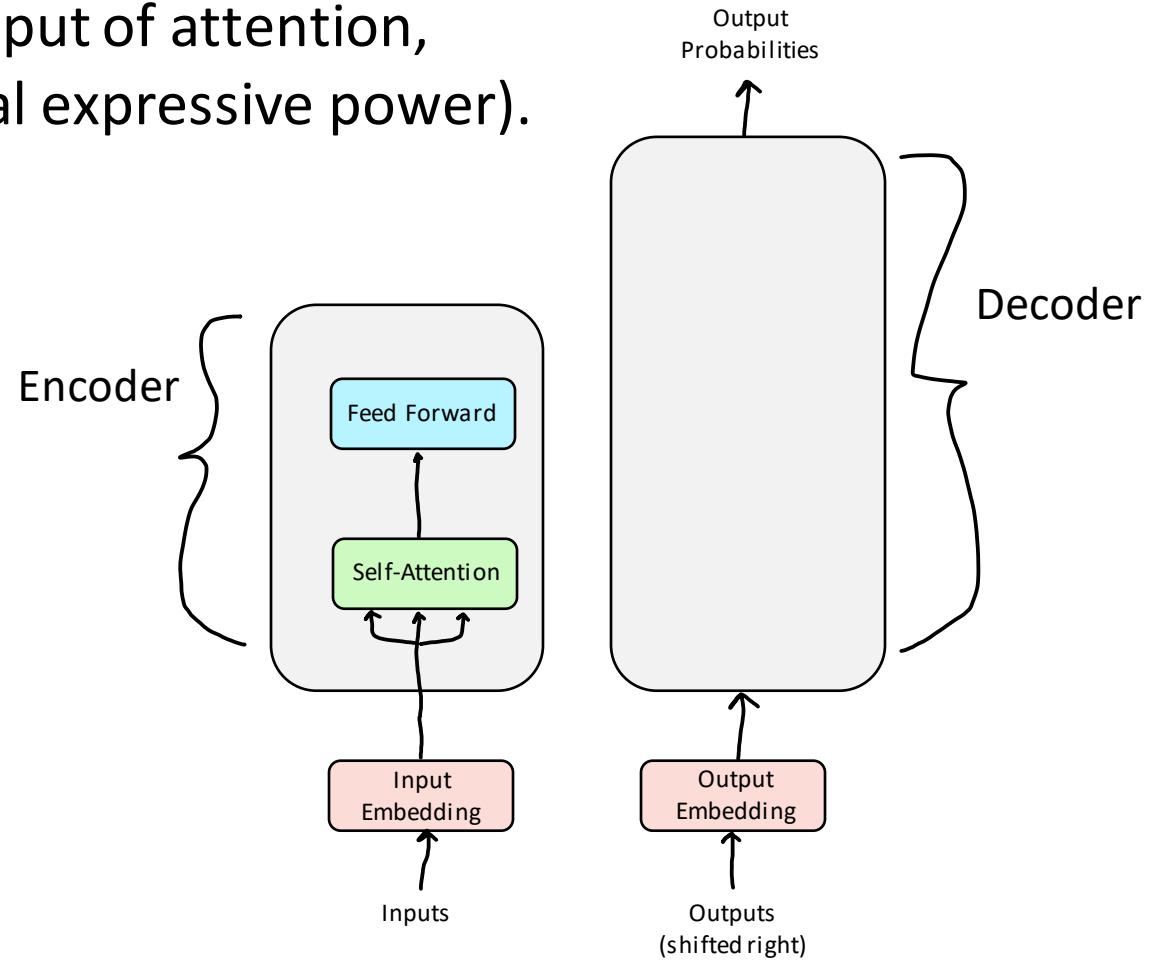
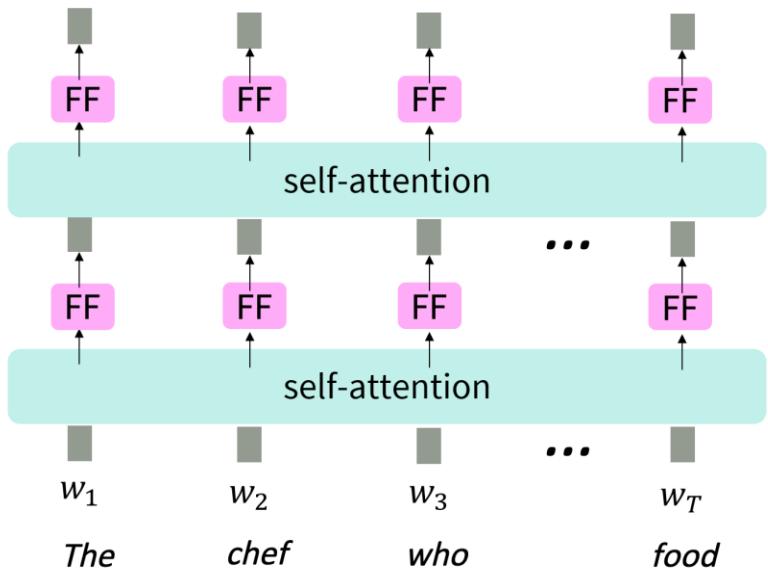


But attention isn't quite all you need!

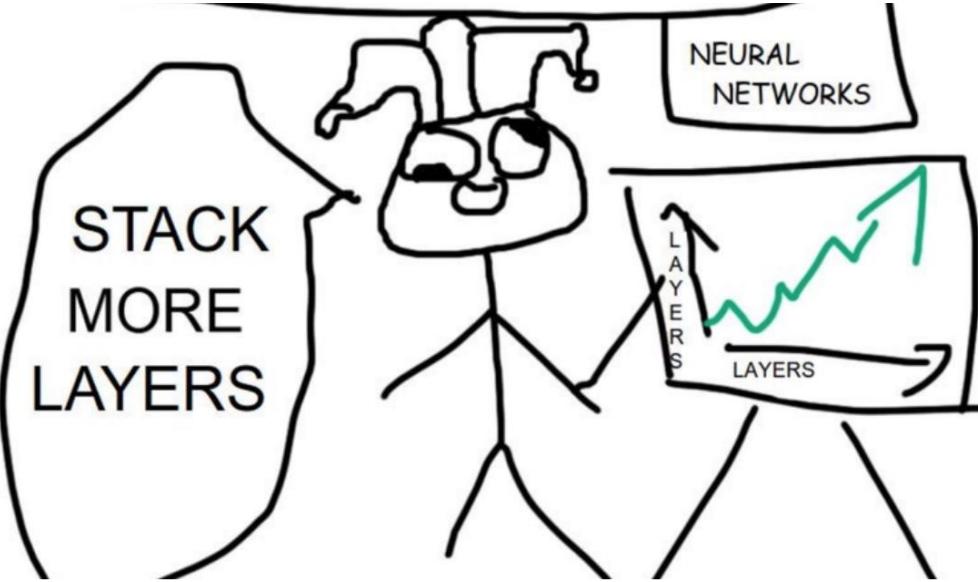
- **Problem:** Since there are no element-wise non-linearities, self-attention is simply performing a re-averaging of the value vectors.
- **Easy fix:** Apply a feedforward layer to the output of attention, providing non-linear activation (and additional expressive power).

Equation for Feed Forward Layer

$$\begin{aligned} m_i &= \text{MLP}(\text{output}_i) \\ &= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2 \end{aligned}$$



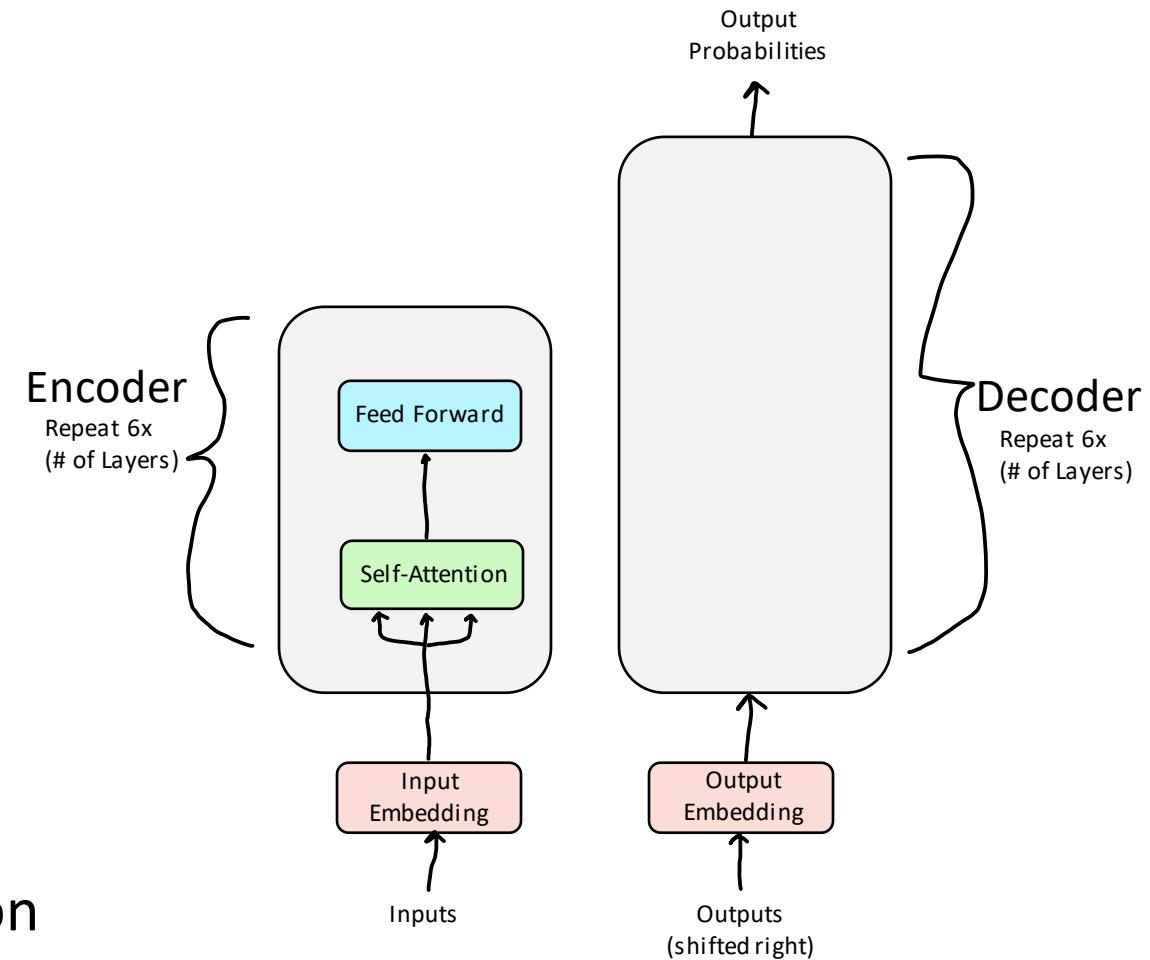
But how do we make this work for deep networks?



Training Trick #1: Residual Connections

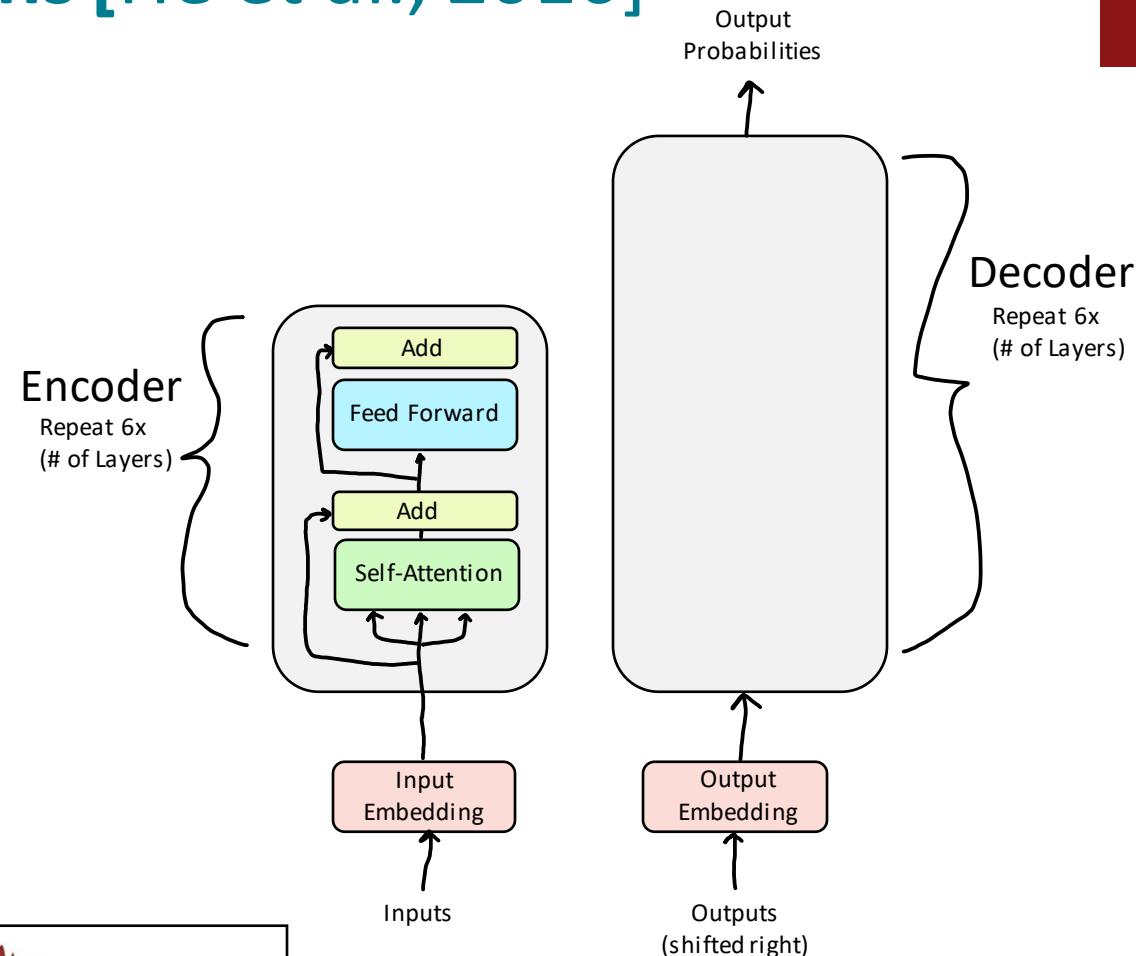
Training Trick #2: LayerNorm

Training Trick #3: Scaled Dot Product Attention

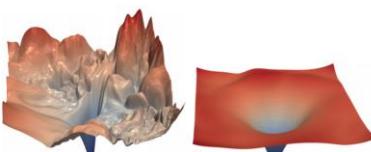


Training Trick #1: Residual Connections [He et al., 2016]

- Residual connections are a simple but powerful technique from computer vision.
- Deep networks are surprisingly bad at learning the identity function!
- Therefore, directly passing "raw" embeddings to the next layer can actually be very helpful!
$$x_\ell = F(x_{\ell-1}) + x_{\ell-1}$$
- This prevents the network from "forgetting" or distorting important information as it is processed by many layers.



Residual connections are also thought to smooth the loss landscape and make training easier!



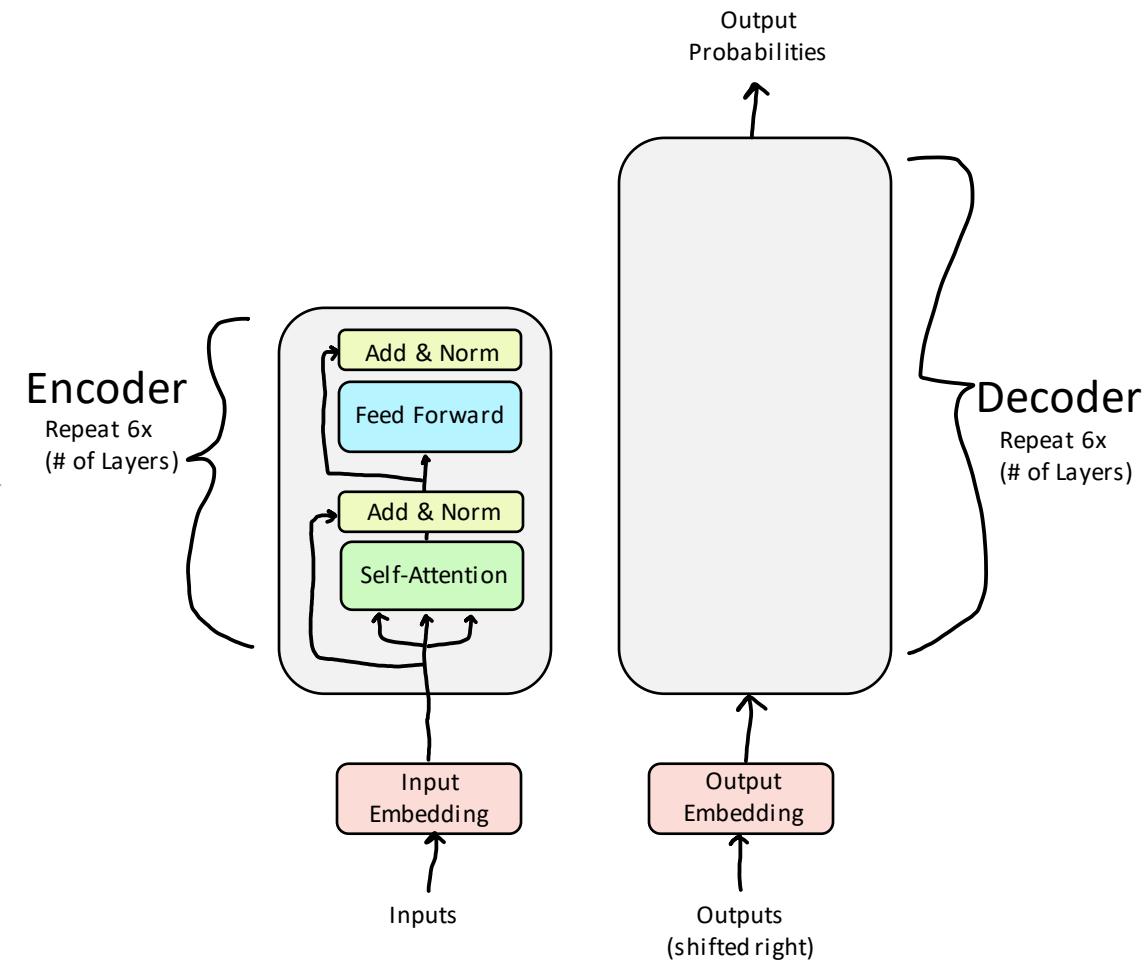
[Loss landscape visualization,
[Li et al., 2018](#), on a ResNet]

Training Trick #2: Layer Normalization [Ba et al., 2016]

- **Problem:** Difficult to train the parameters of a given layer because its input from the layer beneath keeps shifting.
- **Solution:** Reduce uninformative variation by **normalizing** to zero mean and standard deviation of one within each **layer**.

$$\text{Mean: } \mu^l = \frac{1}{H} \sum_{i=1}^H a_i^l \quad \text{Standard Deviation: } \sigma^l = \sqrt{\frac{1}{H} \sum_{i=1}^H (a_i^l - \mu^l)^2}$$

$$x'^{\ell'} = \frac{x^{\ell'} - \mu^{\ell'}}{\sigma^{\ell'} + \epsilon}$$



Training Trick #3: Scaled Dot Product Attention

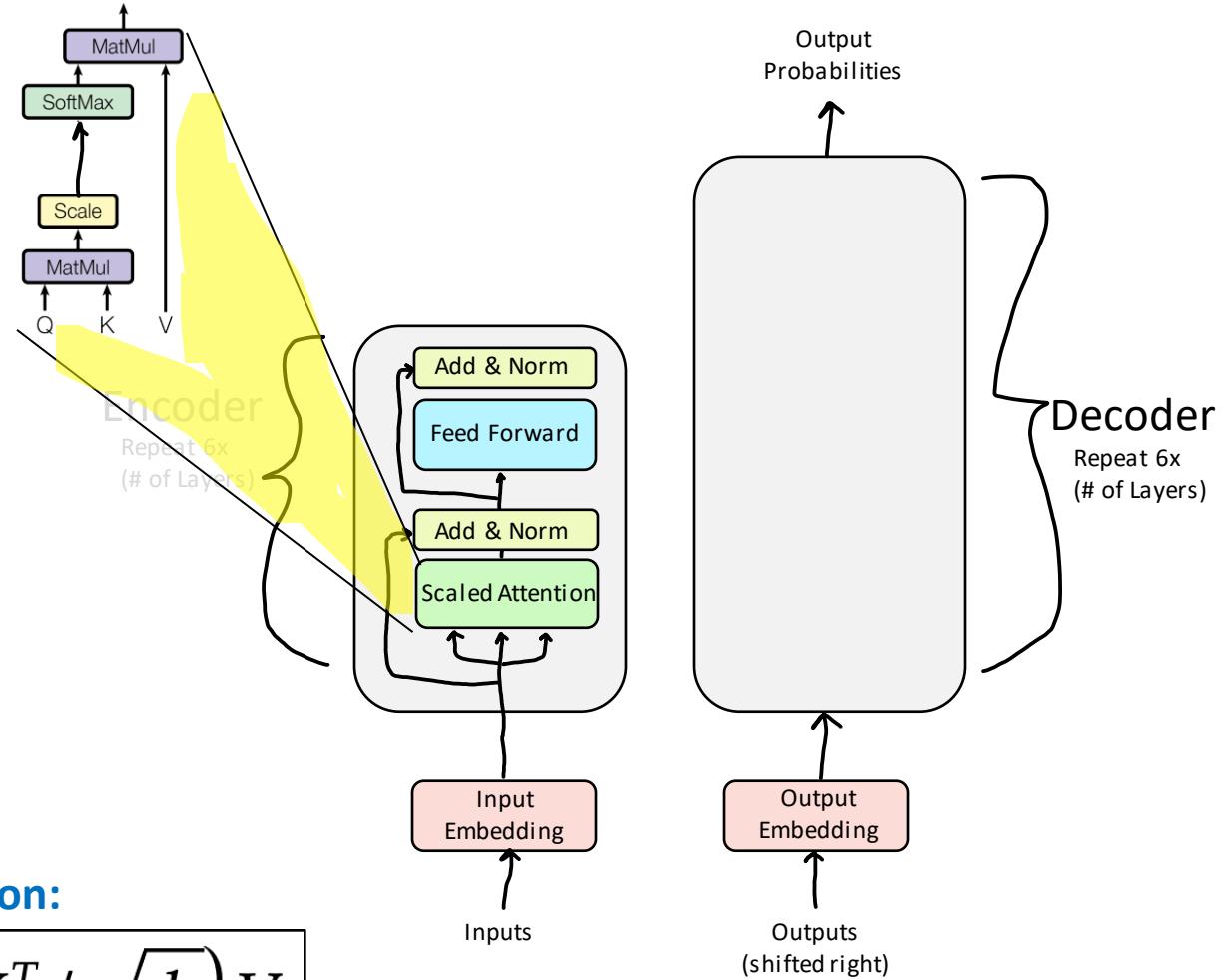
- After LayerNorm, the mean and variance of vector elements is 0 and 1, respectively. (Yay!)
- However, the dot product still tends to take on extreme values, as its variance scales with dimensionality d_k

Quick Statistics Review:

- Mean of sum = sum of means = $d_k * 0 = 0$
- Variance of sum = sum of variances = $d_k * 1 = d_k$
- To set the variance to 1, simply divide by $\sqrt{d_k}$!

Updated Self-Attention Equation:

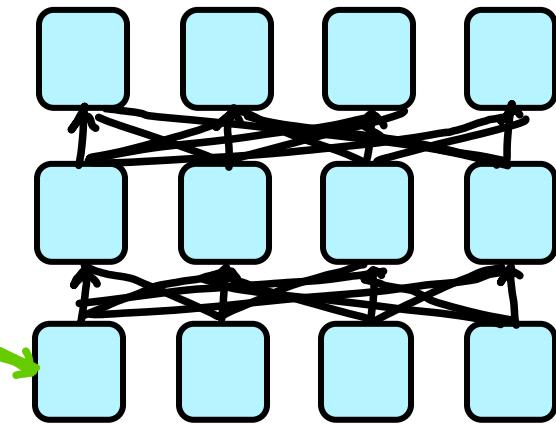
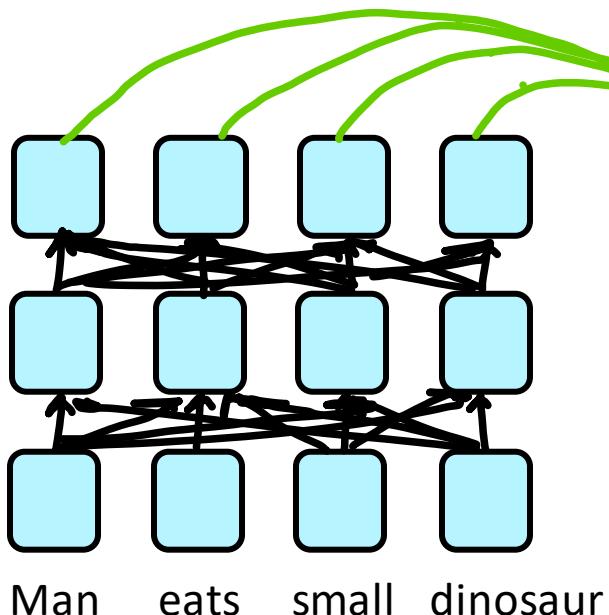
$$Output = \text{softmax}\left(QK^T / \sqrt{d_k}\right)V$$



Major issue!

- We're almost done with the Encoder, but we have a major problem! Has anyone spotted it?
- Consider this sentence:
 - "Man eats small dinosaur."

$$Output = \text{softmax}\left(QK^T / \sqrt{d_k}\right)V$$

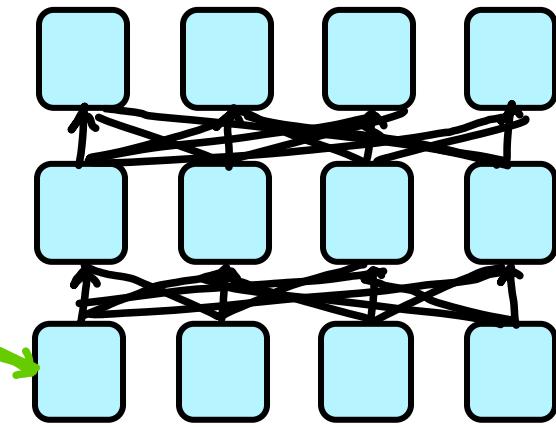
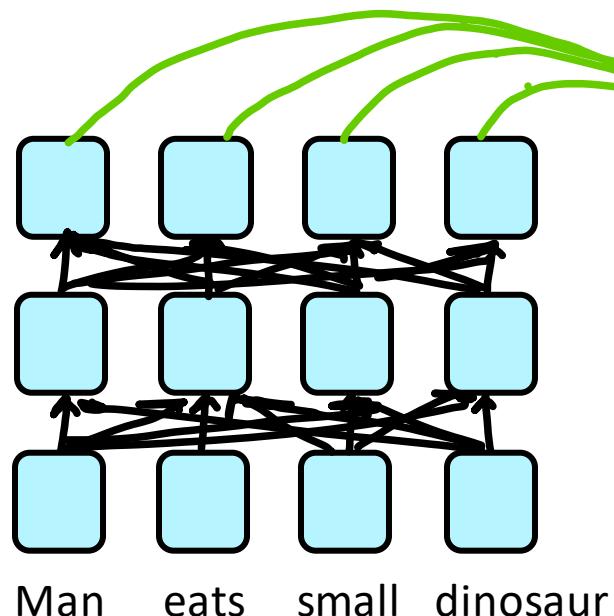


Transformer-Based
Encoder-Decoder Model

Major issue!

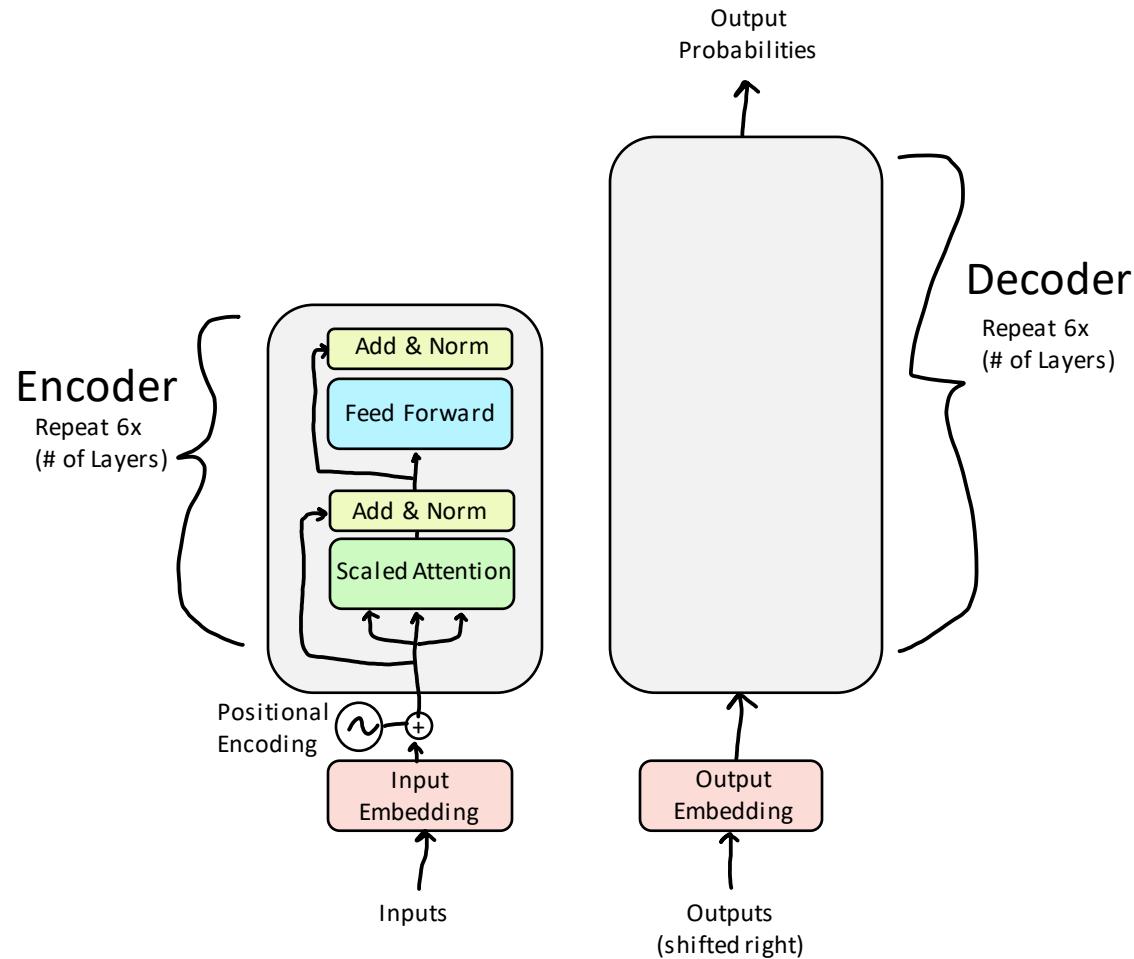
- We're almost done with the Encoder, but we have a major problem! Has anyone spotted it?
- Consider this sentence:
 - "Man eats small dinosaur."
- Wait a minute, order doesn't impact the network at all!
- This seems wrong given that word order does have meaning in many languages, including English!

$$\text{Output} = \text{softmax}\left(QK^T / \sqrt{d_k}\right)V$$



Transformer-Based
Encoder-Decoder Model

Solution: Inject Order Information through Positional Encodings!



Fixing the first self-attention problem: sequence order

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**

$p_i \in \mathbb{R}^d$, for $i \in \{1, 2, \dots, T\}$ are position vectors

- Don't worry about what the p_i are made of yet!
- Easy to incorporate this info into our self-attention block: just add the p_i to our inputs!
- Let $\tilde{v}_i, \tilde{k}_i, \tilde{q}_i$ be our old values, keys, and queries.

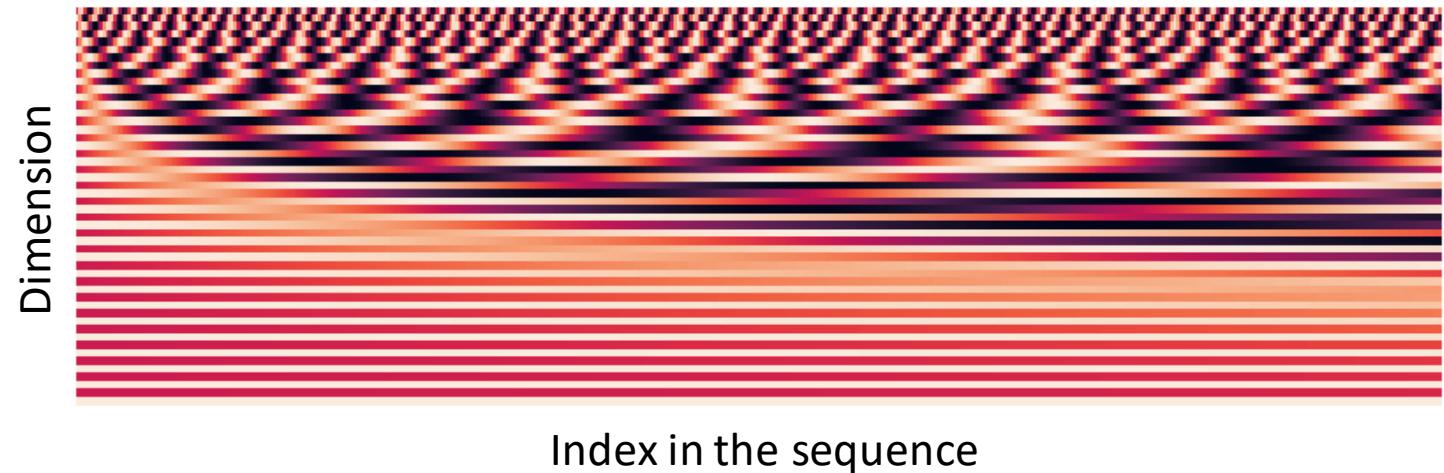
$$\begin{aligned} v_i &= \tilde{v}_i + p_i \\ q_i &= \tilde{q}_i + p_i \\ k_i &= \tilde{k}_i + p_i \end{aligned}$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add...

Position representation vectors through sinusoids

- **Sinusoidal position representations:** concatenate sinusoidal functions of varying periods:

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$

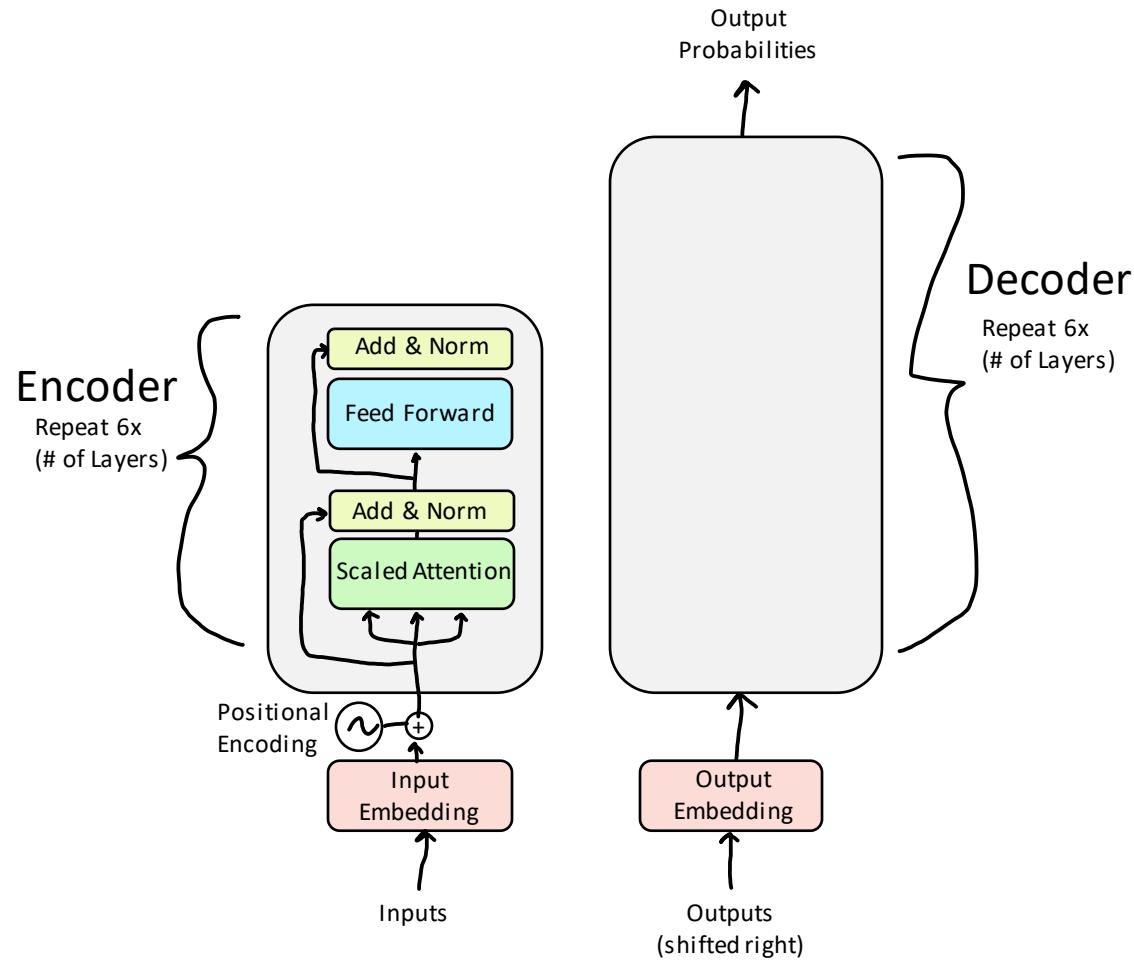


- Pros:
 - Periodicity indicates that maybe “absolute position” isn’t as important
 - Maybe can extrapolate to longer sequences as periods restart
- Cons:
 - Not learnable; also the extrapolation doesn’t really work

Position representation vectors learned from scratch

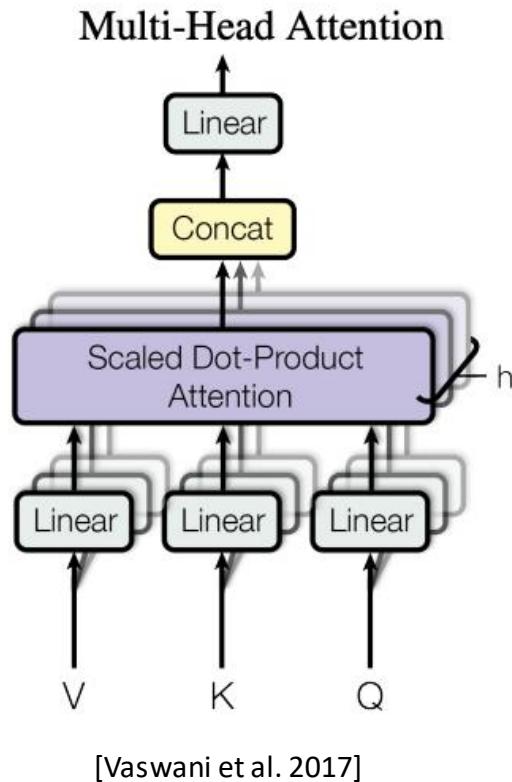
- **Learned absolute position representations:** Let all p_i be learnable parameters!
Learn a matrix $p \in \mathbb{R}^{d \times T}$, *and let each p_i be a column of that matrix!*
- Pros:
 - Flexibility: each position gets to be learned to fit the data
- Cons:
 - Definitely can't extrapolate to indices outside $1, \dots, T$.
- Most systems use this!
- Sometimes people try more flexible representations of position:
 - Relative linear position attention [\[Shaw et al., 2018\]](#)
 - Dependency syntax-based position [\[Wang et al., 2019\]](#)

Solution: Inject Order Information through Positional Encodings!



Multi-Headed Self-Attention: k heads are better than 1!

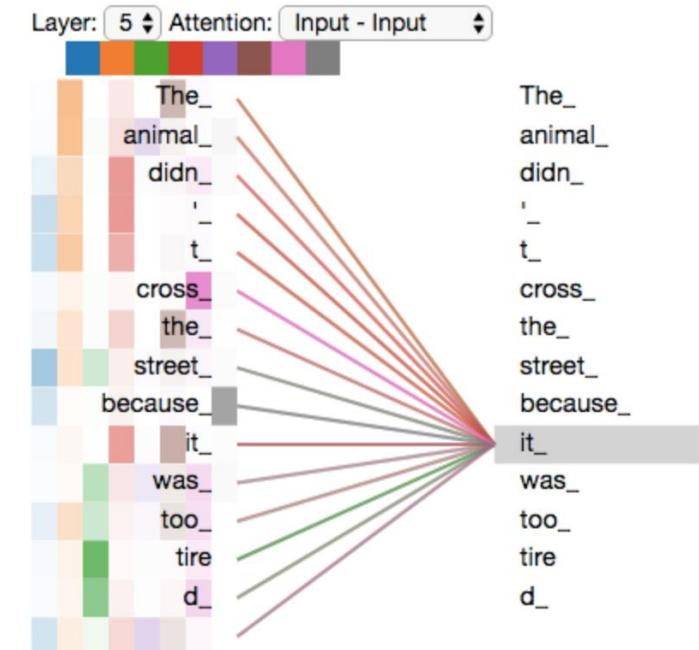
- **High-Level Idea:** Let's perform self-attention multiple times in parallel and combine the results.



Wizards of the Coast, Artist: Todd Lockwood

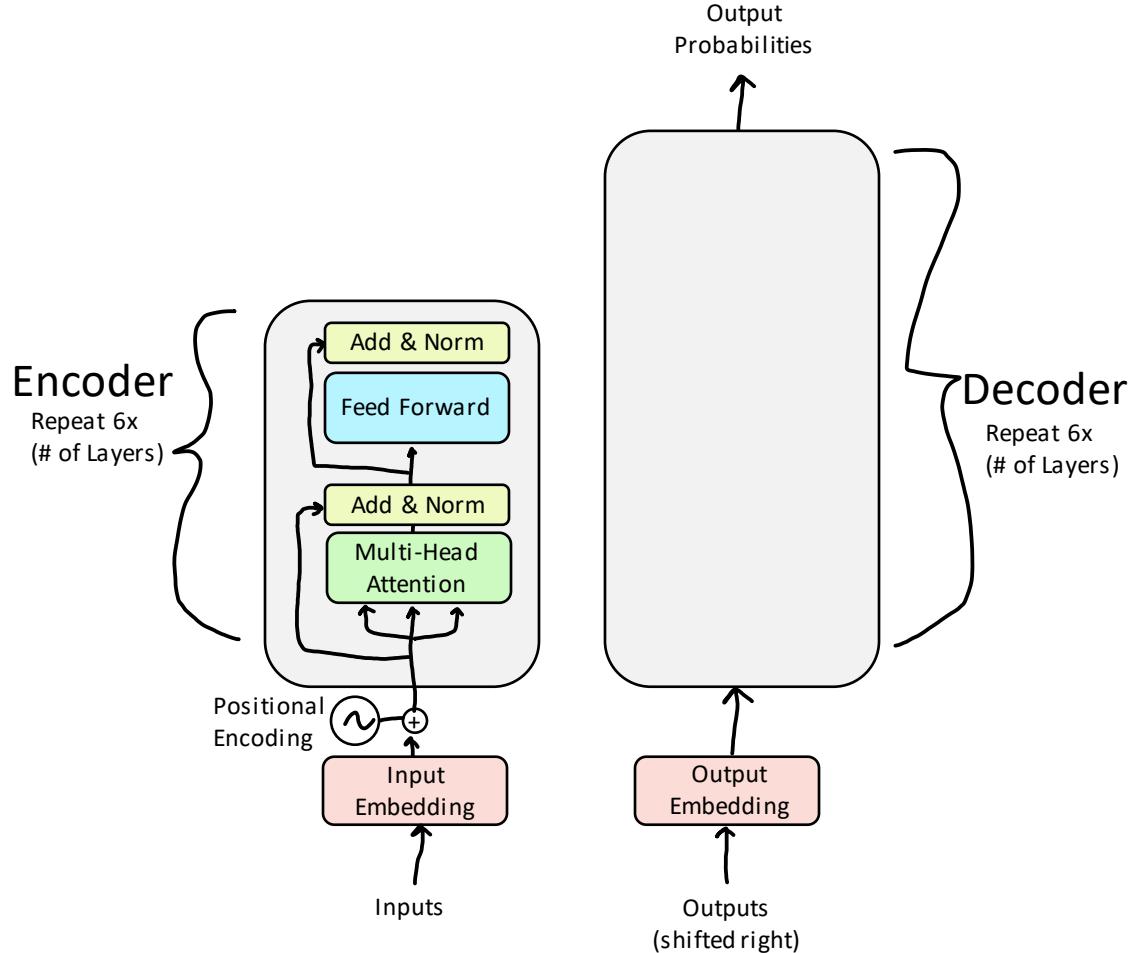
The Transformer Encoder: Multi-headed Self-Attention

- What if we want to look in multiple places in the sentence at once?
 - For word i , self-attention “looks” where $x_i^T Q^T K x_j$ 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:
 - $\text{output}_\ell = \text{softmax}(X Q_\ell K_\ell^T X^T) * X V_\ell$, where $\text{output}_\ell \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
 - $\text{output} = Y[\text{output}_1; \dots; \text{output}_h]$, where $Y \in \mathbb{R}^{d \times d}$



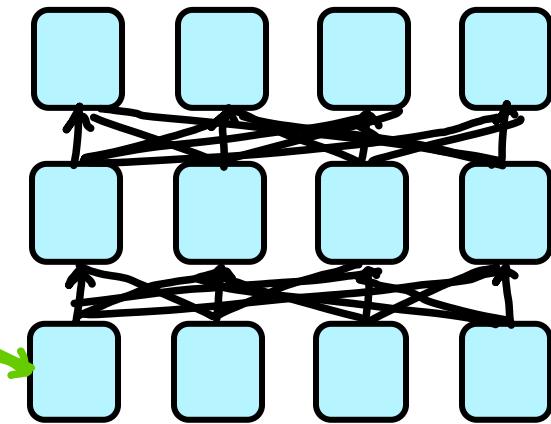
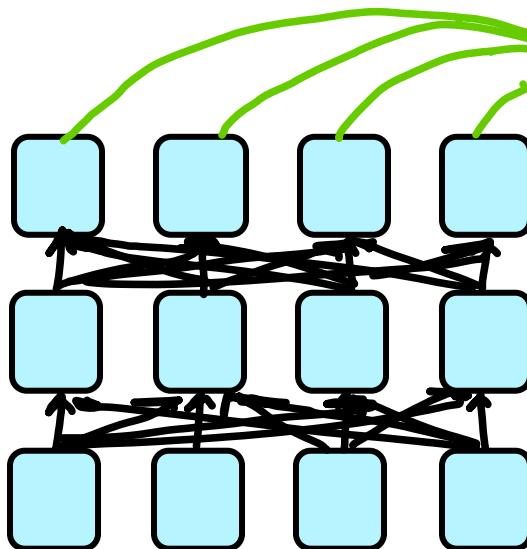
Credit to <https://jalammar.github.io/illustrated-transformer/>

Yay, we've completed the Encoder! Time for the Decoder...



Decoder: Masked Multi-Head Self-Attention

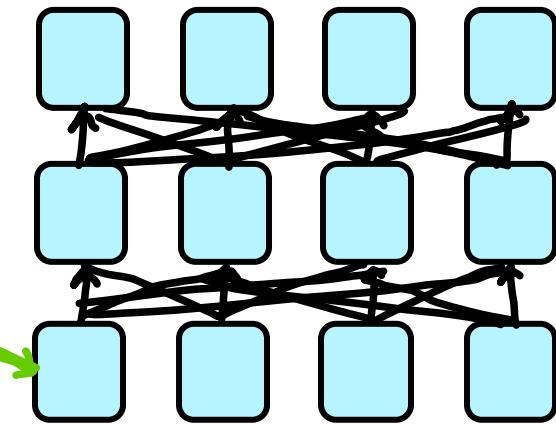
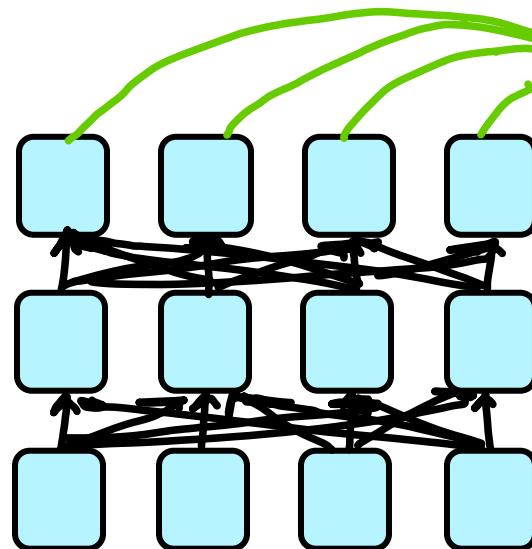
- **Problem:** How do we keep the decoder from cheating? If we have a language modeling objective, can't the network just look ahead and "see" the answer?



Transformer-Based
Encoder-Decoder Model

Decoder: Masked Multi-Head Self-Attention

- **Problem:** How do we keep the decoder from "cheating"? If we have a language modeling objective, can't the network just look ahead and "see" the answer?
- **Solution:** Masked Multi-Head Attention. At a high-level, we hide (mask) information about future tokens from the model.



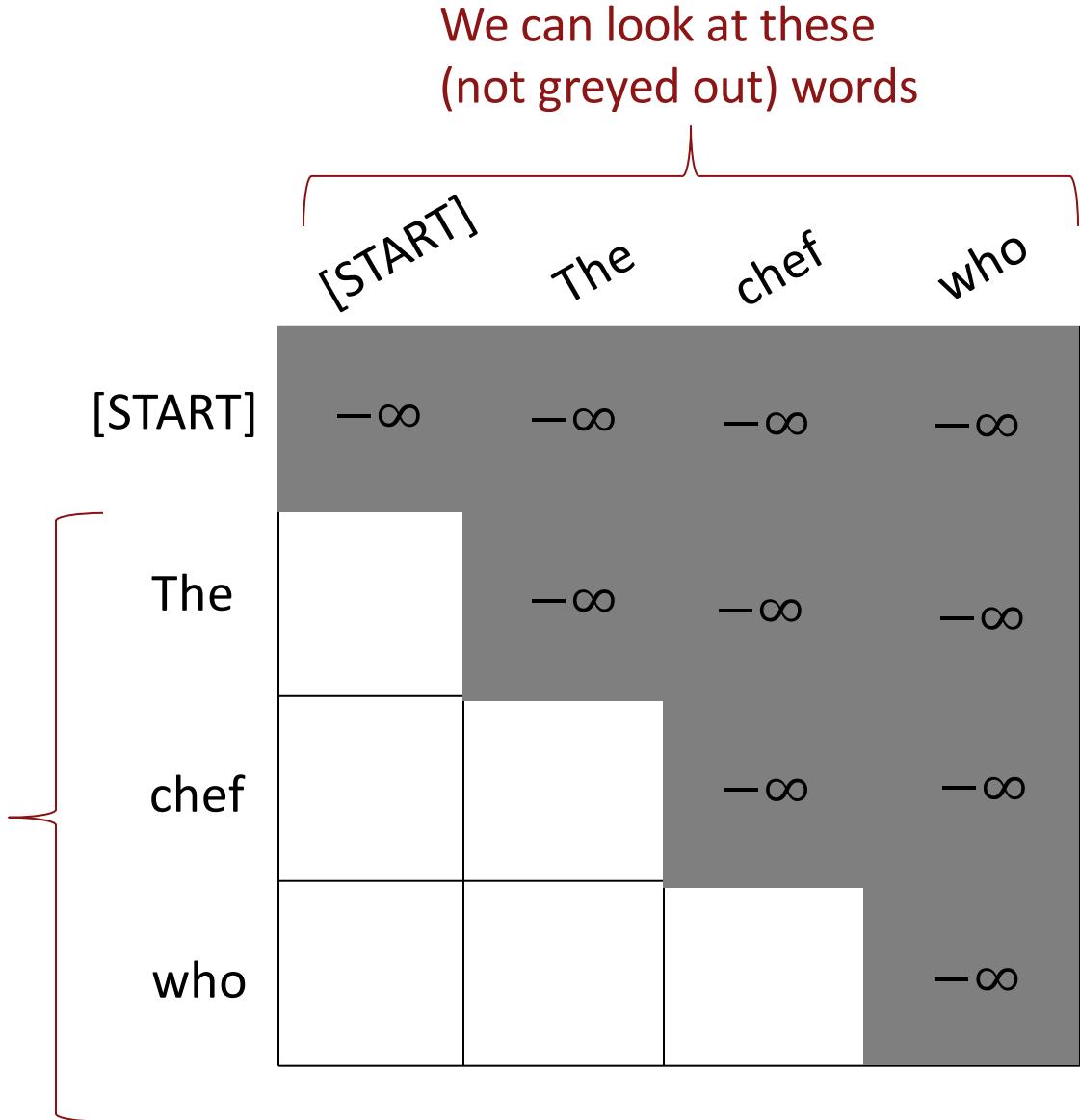
Transformer-Based
Encoder-Decoder Model

Masking the future in self-attention

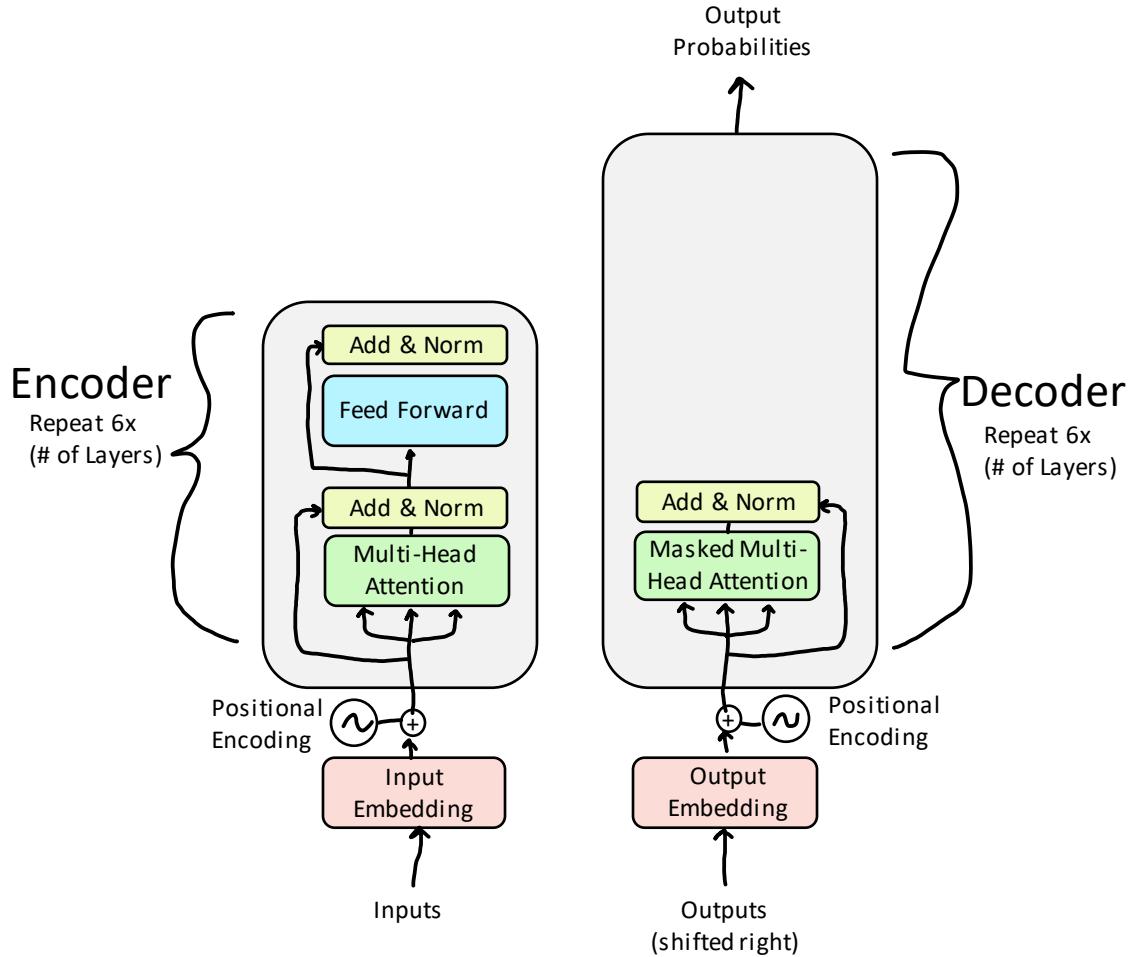
- To use self-attention in **decoders**, we need to ensure we can't peek at the future.
- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^T k_j, & j < i \\ -\infty, & j \geq i \end{cases}$$

For encoding
these words

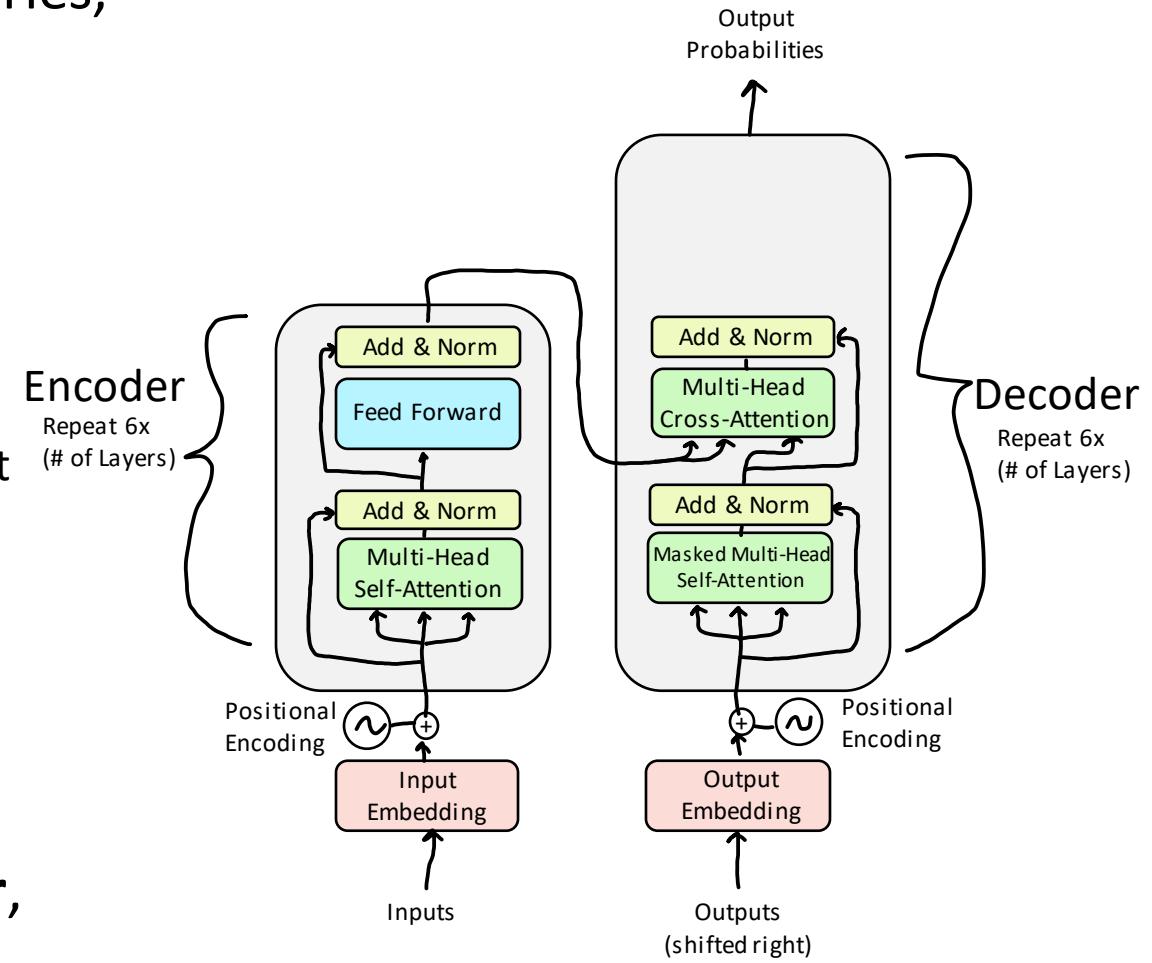


Decoder: Masked Multi-Headed Self-Attention

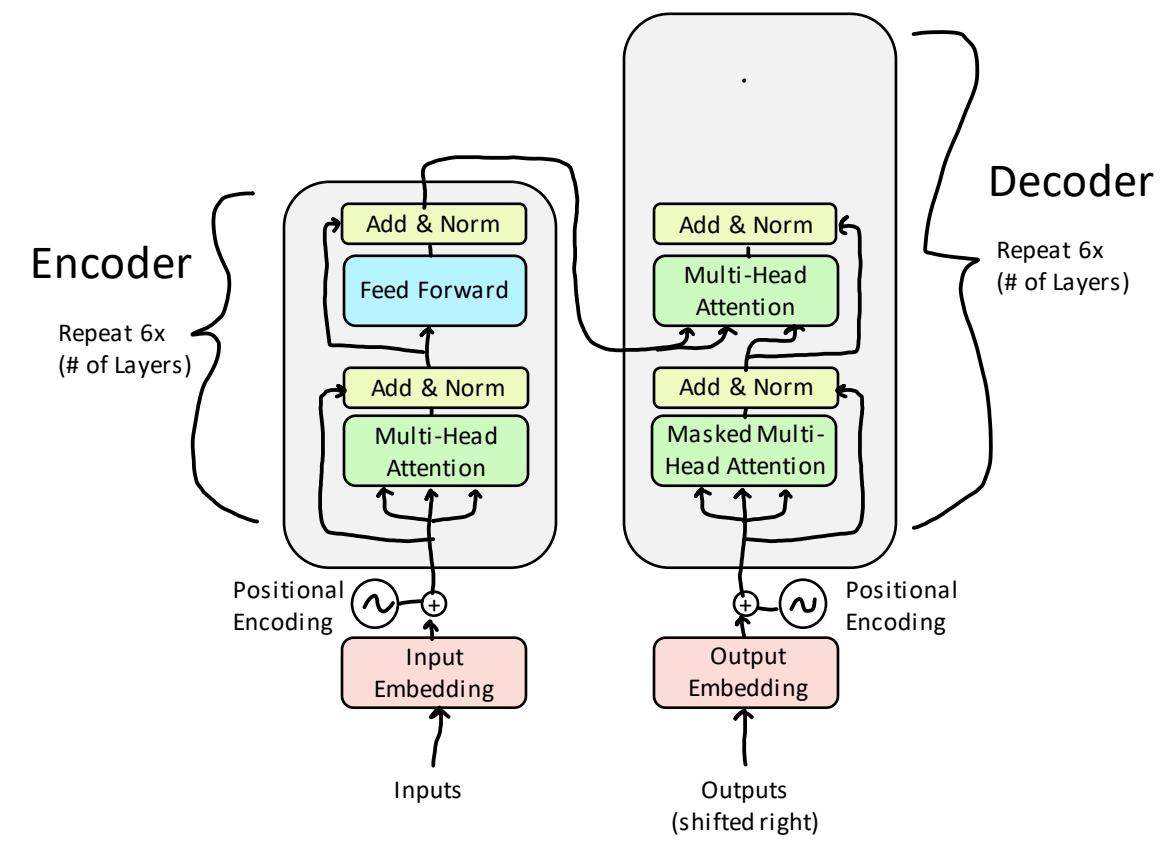


Encoder-Decoder Attention

- 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_T be **output vectors from the Transformer encoder**; $x_i \in \mathbb{R}^d$
- Let z_1, \dots, z_T be input vectors from the ^{Click to add text}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**, $q_i = Qz_i$.

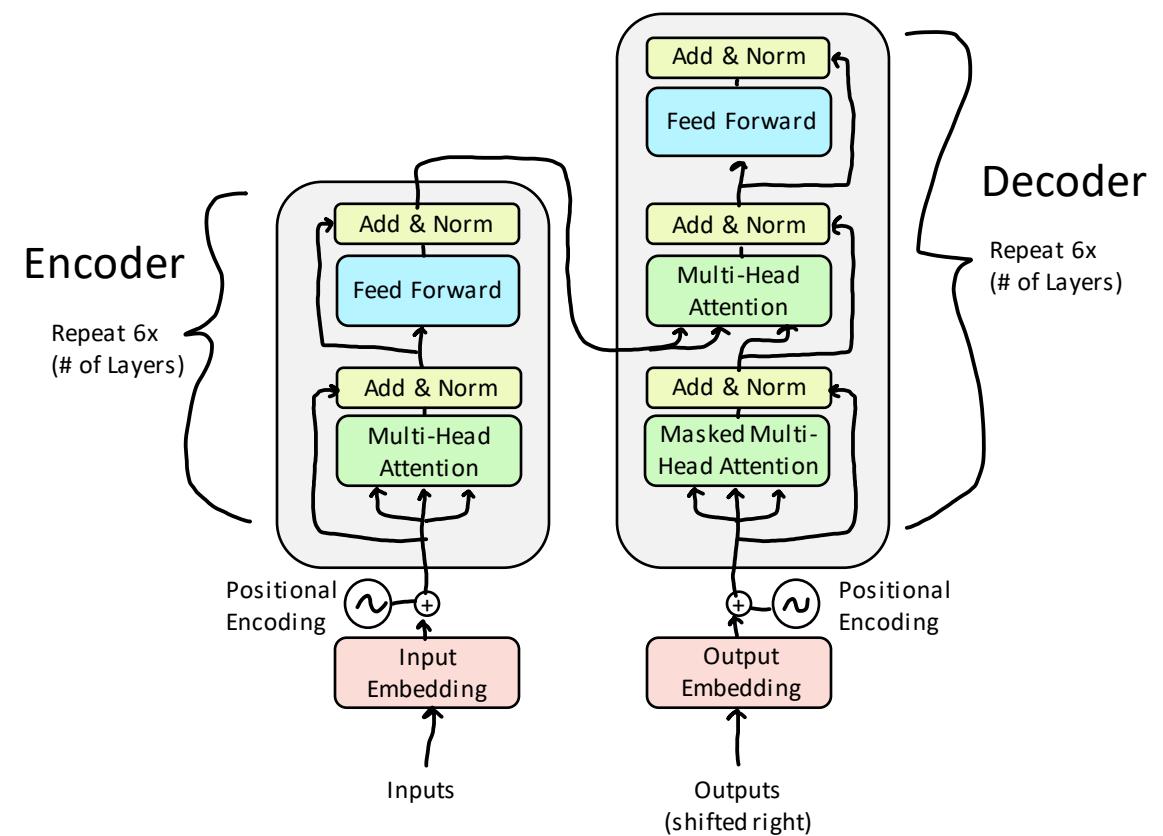


Decoder: Finishing touches!



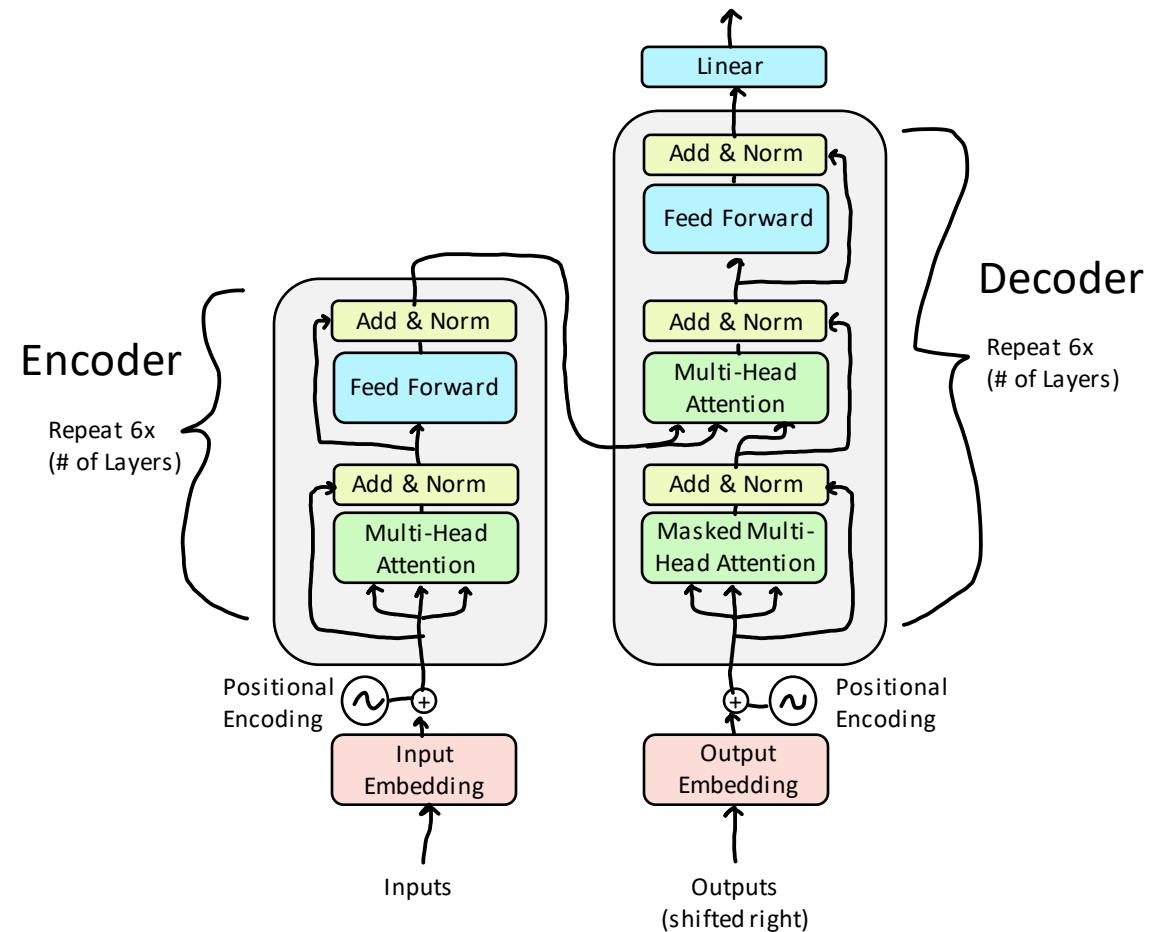
Decoder: Finishing touches!

- Add a feed forward layer (with residual connections and layer norm)



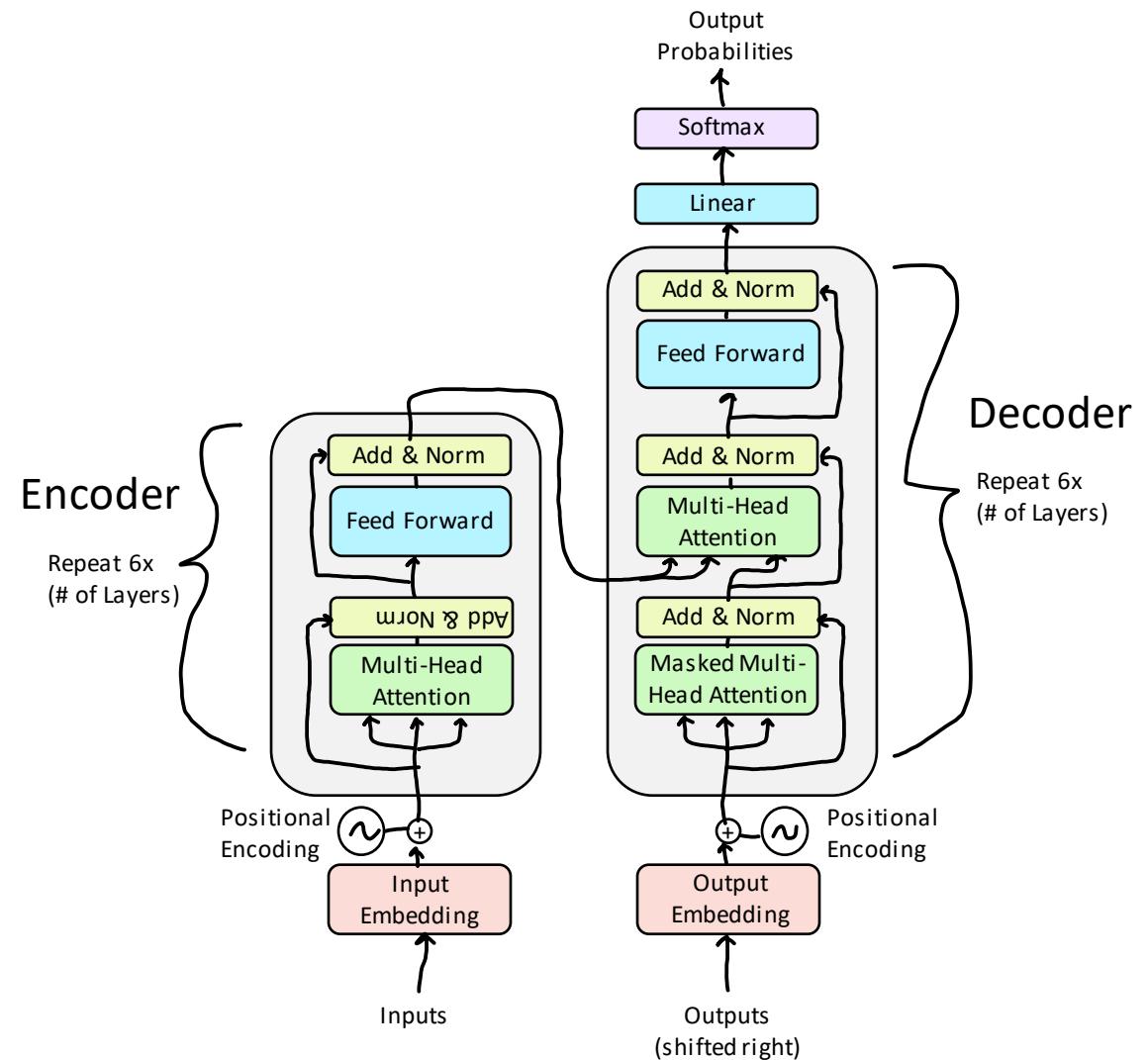
Decoder: Finishing touches!

- Add a feed forward layer (with residual connections and layer norm)
- Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits)

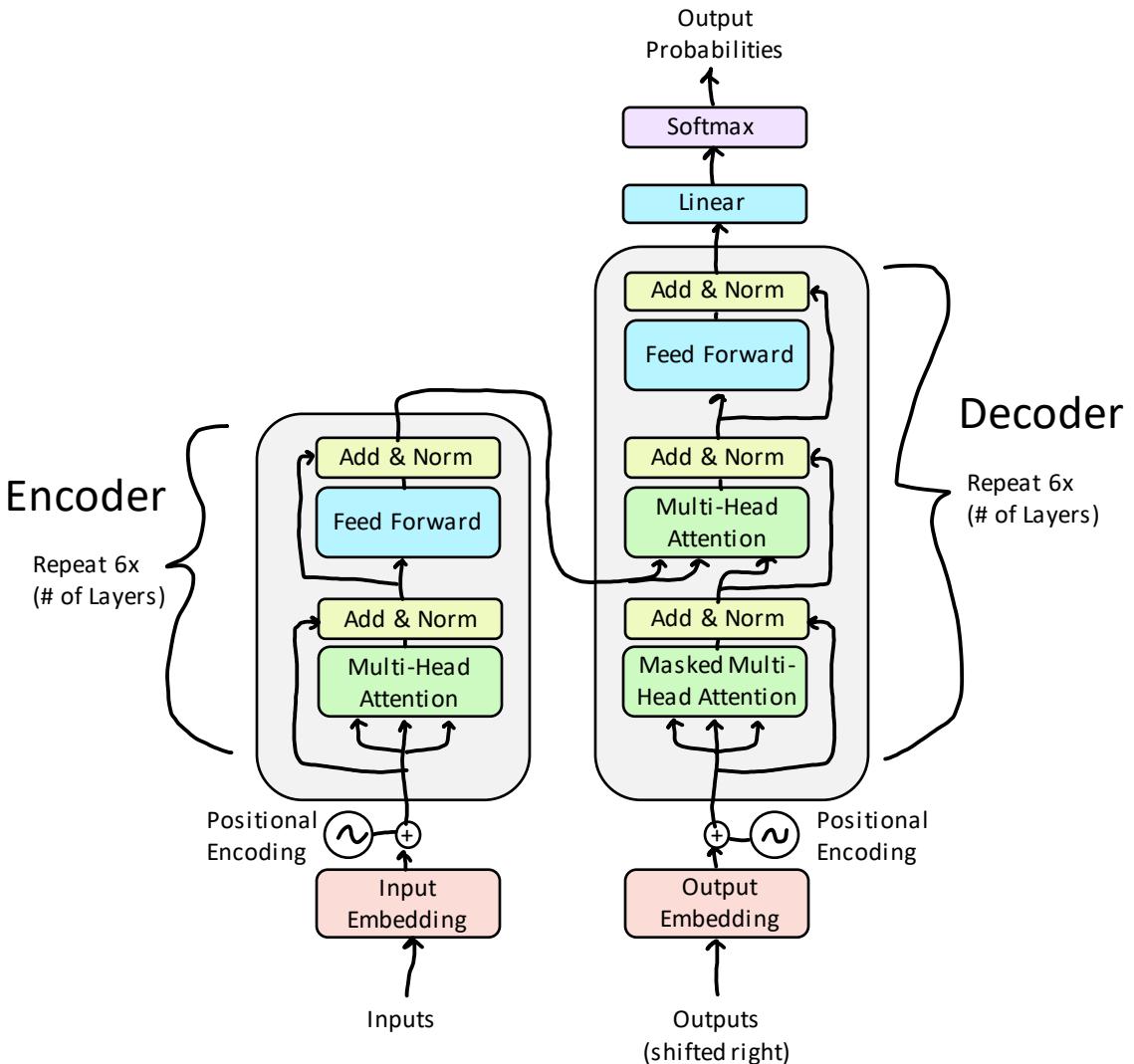


Decoder: Finishing touches!

- Add a feed forward layer (with residual connections and layer norm)
- Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits)
- Add a final softmax to generate a probability distribution of possible next words!



Recap of Transformer Architecture



Outline

- 1.** Impact of Transformers on NLP (and ML more broadly)
- 2.** From Recurrence (RNNs) to Attention-Based NLP Models
- 3.** Understanding the Transformer Model
- 4.** Drawbacks and Variants of Transformers

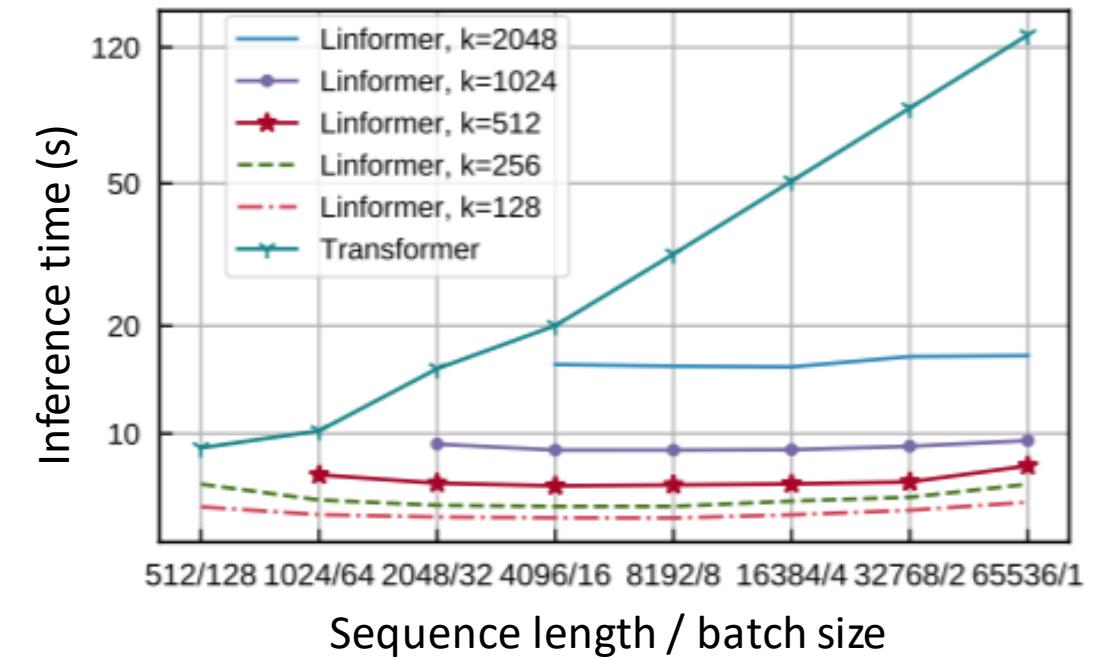
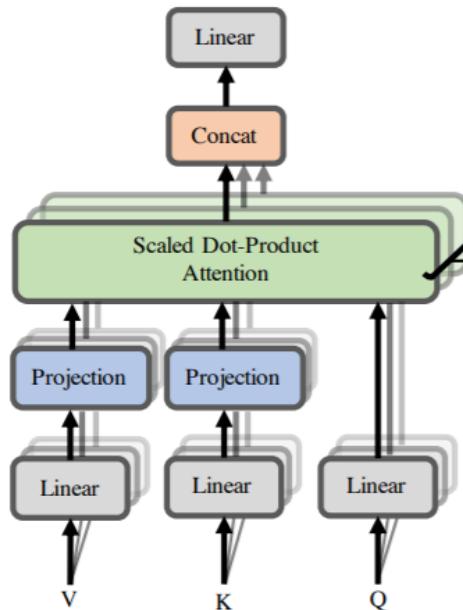
What would we like to fix about the Transformer?

- **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?
 - Relative linear position attention [\[Shaw et al., 2018\]](#)
 - Dependency syntax-based position [\[Wang et al., 2019\]](#)

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](#)]

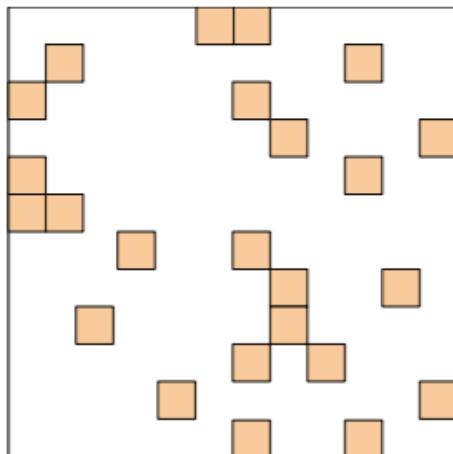
Key idea: map the sequence length dimension to a lower-dimensional space for values, keys



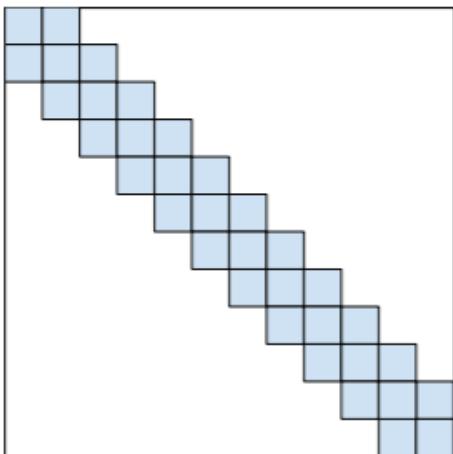
Recent work on improving on quadratic self-attention cost

- 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](#)]

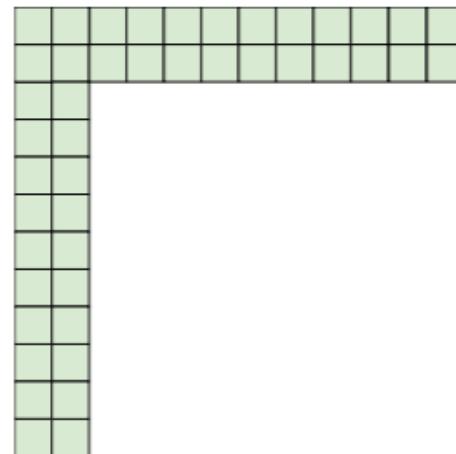
Key idea: replace all-pairs interactions with a family of other interactions, **like local windows, looking at everything, and random interactions.**



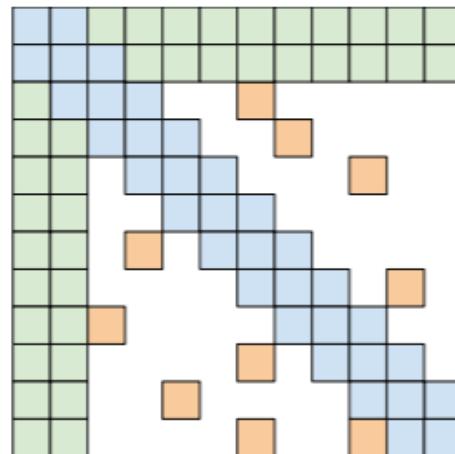
(a) Random attention



(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	11.17T	3.50	2.182 ± 0.005	1.838	71.66	17.78	23.02	26.62
GeLU	223M	11.17T	3.58	2.179 ± 0.003	1.838	75.79	17.86	25.13	26.47
Swish	223M	11.17T	3.62	2.186 ± 0.003	1.847	73.77	17.74	24.34	26.75
ELU	223M	11.17T	3.57	2.186 ± 0.003	1.845	67.35	20.02	26.08	
GLU	223M	11.17T	3.59	2.174 ± 0.003	1.814	74.20	17.42	24.34	21.12
GeGLU	223M	11.17T	3.55	2.130 ± 0.006	1.792	75.96	18.27	24.87	26.57
ReLU	223M	11.17T	3.57	2.145 ± 0.004	1.803	76.17	18.36	24.87	27.02
SeLU	223M	11.17T	3.55	2.115 ± 0.004	1.948	68.76	16.76	22.75	25.99
SwiGLU	223M	11.17T	3.53	2.127 ± 0.003	1.789	76.00	18.20	24.34	27.02
LoGLU	223M	11.17T	3.50	3.110 ± 0.006	1.900	72.17	18.24	24.34	26.53
Sigmoid	223M	11.17T	3.63	2.291 ± 0.019	1.867	74.31	17.51	23.02	26.30
Softplus	223M	11.17T	3.47	2.207 ± 0.011	1.850	72.45	17.65	24.34	26.89
RMS Norm	223M	11.17T	3.68	2.167 ± 0.008	1.821	75.45	17.94	24.07	27.14
Rezero	223M	11.17T	3.51	2.282 ± 0.003	1.939	61.69	15.64	20.90	26.37
Rezero + LayerNorm	223M	11.17T	3.26	2.223 ± 0.006	1.858	70.42	17.58	23.02	26.29
Rezero + RMS Norm	223M	11.17T	3.34	2.221 ± 0.009	1.875	70.33	17.32	23.02	26.19
Fixup	223M	11.17T	2.95	2.382 ± 0.012	2.067	58.56	14.42	23.02	26.31
24 layers, $d_g = 1536$, $H = 6$	224M	11.17T	3.33	2.290 ± 0.007	1.843	74.89	17.75	25.13	26.89
18 layers, $d_g = 2048$, $H = 8$	223M	11.17T	3.38	2.185 ± 0.005	1.831	76.45	16.83	24.34	27.10
8 layers, $d_g = 4096$, $H = 18$	223M	11.17T	3.69	2.190 ± 0.005	1.847	74.58	17.69	23.28	26.85
6 layers, $d_g = 6144$, $H = 24$	223M	11.17T	3.70	2.200 ± 0.007	1.857	73.55	17.59	24.60	26.66
Block diagonal	63M	11.17T	3.91	2.407 ± 0.037	2.064	61.90	14.53	21.96	25.48
+ Factorized embeddings	15.5M	9.47T	4.21	2.431 ± 0.035	2.183	60.84	14.00	19.84	25.27
+ Factorized & shared embeddings	20.0M	9.17T	4.37	2.907 ± 0.313	2.385	53.95	11.37	19.84	25.19
Encoder only block sharing	170M	11.17T	3.68	2.298 ± 0.023	1.929	69.60	16.23	23.02	26.23
Decoder only block sharing	144M	11.17T	3.70	2.352 ± 0.029	2.082	67.93	16.13	23.81	26.08
Factorized Embedding	227M	9.47T	3.80	2.208 ± 0.006	1.855	70.41	15.92	22.75	26.50
Factorized & shared embeddings	202M	9.17T	3.92	2.320 ± 0.010	1.952	68.69	16.33	22.22	26.44
Tied encoder/decoder input embeddings	248M	11.17T	3.55	2.192 ± 0.002	1.840	71.70	17.72	24.34	26.49
Tied decoder input and output embeddings	248M	11.17T	3.57	2.187 ± 0.007	1.827	74.86	17.74	24.87	26.67
Untied embeddings	273M	11.17T	3.53	2.195 ± 0.005	1.834	72.99	17.58	23.28	26.48
Adaptive input embeddings	204M	9.27T	3.55	2.290 ± 0.002	1.899	66.57	16.21	24.07	26.66
Adaptive softmax	204M	9.27T	3.60	2.364 ± 0.005	1.982	72.91	16.67	21.16	25.56
Adaptive softmax without projection	223M	10.87T	3.43	2.229 ± 0.009	1.914	71.82	17.10	23.02	25.72
Mixture of softmaxes	232M	16.37T	2.24	2.227 ± 0.017	1.821	76.77	17.62	22.75	26.82
Transparent attention	223M	11.17T	3.33	2.181 ± 0.014	1.874	54.31	10.40	21.16	26.80
Dynamic convolution	257M	11.87T	2.65	2.403 ± 0.009	2.047	58.30	12.67	21.16	17.03
Lightweight convolution	224M	10.47T	4.07	2.370 ± 0.010	1.989	63.07	14.86	23.02	24.73
Evolved Transformer	217M	9.97T	3.09	2.220 ± 0.003	1.863	73.67	10.76	24.07	26.58
Synthesizer (dense)	224M	11.47T	3.47	2.334 ± 0.021	1.962	61.03	14.27	16.14	26.63
Synthesizer (dense plus)	243M	12.67T	3.22	2.191 ± 0.010	1.840	73.98	16.96	23.81	26.71
Synthesizer (dense plus alpha)	243M	12.67T	3.01	2.180 ± 0.007	1.828	74.25	17.02	23.28	26.61
Synthesizer (factorized)	207M	10.17T	3.94	2.341 ± 0.017	1.968	62.78	15.39	23.55	26.42
Synthesizer (random)	254M	10.17T	4.08	2.326 ± 0.012	2.009	54.27	10.35	19.56	26.44
Synthesizer (random plus)	292M	12.07T	3.63	2.189 ± 0.004	1.842	73.32	17.04	24.87	26.43
Synthesizer (random plus alpha)	292M	12.07T	3.42	2.186 ± 0.007	1.828	75.24	17.08	24.08	26.39
Universal Transformer	84M	40.07T	0.88	2.406 ± 0.036	2.053	70.13	14.09	19.05	23.91
Mixture of experts	648M	11.77T	3.20	2.148 ± 0.006	1.785	74.55	18.13	24.08	26.94
Switch Transformer	1100M	11.77T	3.18	2.135 ± 0.007	1.758	75.38	18.02	26.19	26.81
Funnel Transformer	223M	1.97T	4.30	2.288 ± 0.008	1.918	67.34	16.26	22.75	23.20
Weighted Transformer	280M	71.07T	0.59	2.378 ± 0.021	1.989	69.04	16.98	23.02	26.30
Product key memory	421M	386.67T	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†

Parting remarks

- Yay, you now understand Transformers!
- Next class, we will see how pre-training can take performance to the next level!
- Good luck on assignment 4!
- Remember to work on your project proposal!