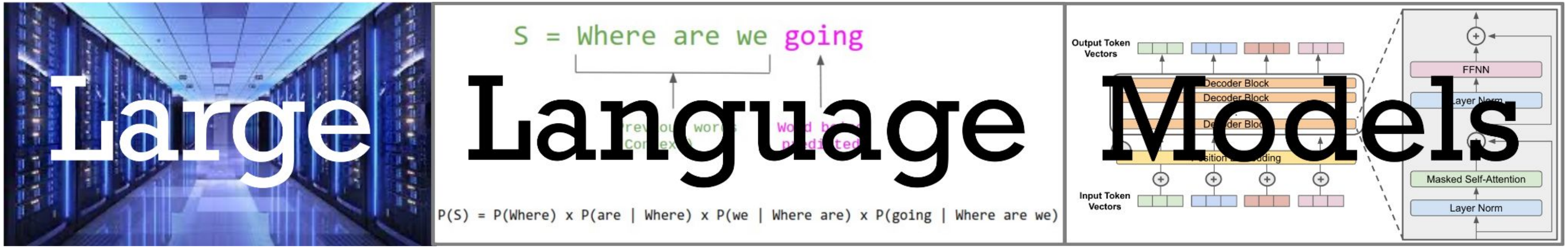


CIS 7000 - Fall 2024

The Transformer Architecture: Part I

Professor Mayur Naik

Slides largely reused from Stanford's CS224N: Natural Language Processing with Deep Learning (Spring'24).

Recap of Last Lecture

The Pre-Transformer Era: Recurrent Neural Networks (RNNs)

Basic architecture

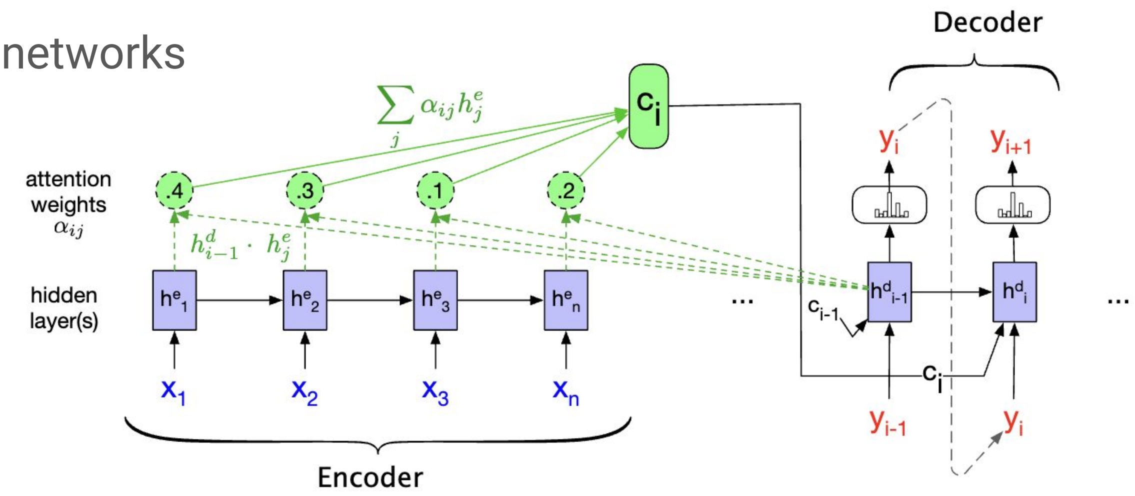
Advantages over feed-forward networks

Variations and applications

Remaining limitations

Seq2Seq architecture

Attention mechanism



Many Interesting Questions!

How many layers to use for the encoder RNN vs. the decoder RNN for a machine translation task from given source language to a given target language?

How to set the incoming hidden state $\mathbf{h}_0$ of the first RNN unit in different RNN architectures?

How to incorporate context $\mathbf{c}_i$ computed from RNN encoder's hidden states into the i^{th} step of the RNN decoder's computation?

$$\mathbf{h}_i^d = g(\mathbf{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

What exactly is the attention mechanism that was introduced in Bahdanau et al.'s ICLR 2015 paper titled "Neural Machine Translation by Jointly Learning to Align and Translate"? Was it learned?

Homework 1 Due Dates

Part I is due this Wednesday (**Sept 18**).

Part II is due next Wednesday (**Sept 25**).

Part III will be released this Wednesday and will be due the following Wednesday (**Oct 2**).

All deadlines are **11:59 pm ET**. No extensions!

Calendars, Gradescope, etc. to be updated soon.

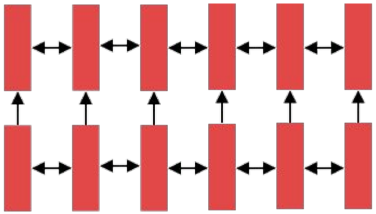
Today’s Agenda

- Impact of Transformers
- From Recurrence (RNNs) to Attention-Based Models
- The Transformer Block

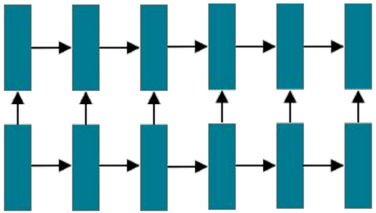
Impact of Transformers

The Story So Far: RNNs for (Most) NLP

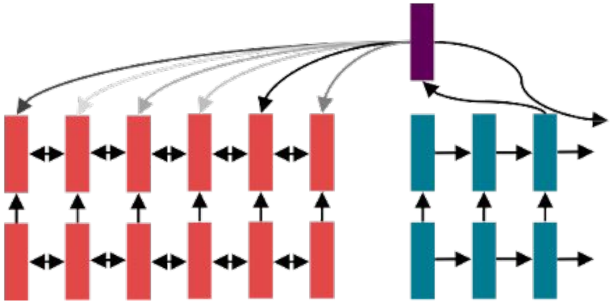
- Circa 2016, the de facto strategy in NLP is to encode sentences with a bidirectional LSTM (e.g., the source sentence in a translation).



- Define your output (parse, sentence, summary) as a sequence, and use an LSTM to generate it.



- Use attention to allow flexible access to memory.



Attention Is All You Need

We saw that attention dramatically improves the performance of RNNs.

Transformers take this idea one step further!

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

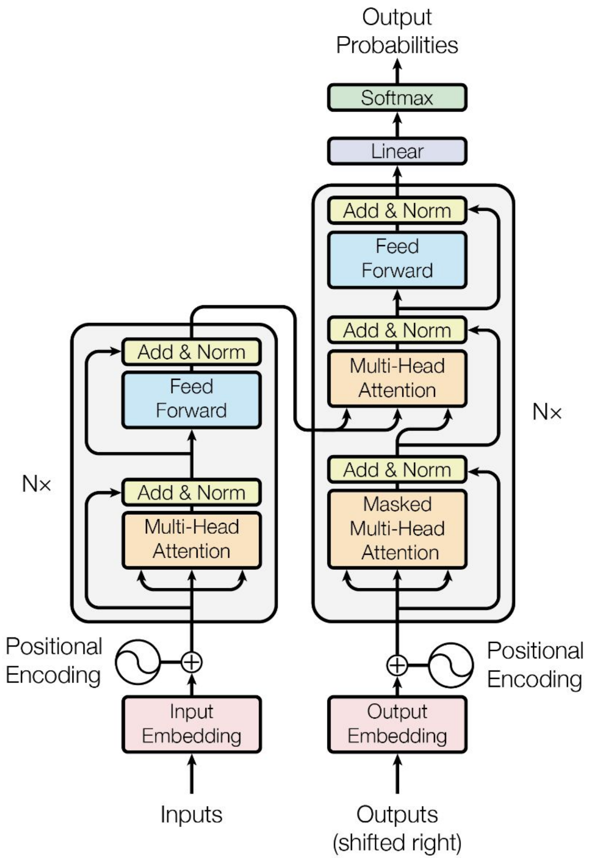
Jakob Uszkoreit* Google Research usz@google.com

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



Great Results With Transformers As Chatbots

Transformer-based models dominate the LMSYS Chatbot Arena Leaderboard!

Rank* (UB)	Model	Arena Score	95% CI	Votes	Organization	License	Knowledge Cutoff
1	ChatGPT-4o-latest (2024-08-08)	1316	+4/-3	31148	OpenAI	Proprietary	2023/10
2	Gemini-1.5-Pro-Exp-0827	1300	+4/-4	22844	Google	Proprietary	2023/11
2	Grok-2-08-13	1294	+4/-4	16215	xAI	Proprietary	2024/3
4	GPT-4o-2024-05-13	1285	+3/-2	86306	OpenAI	Proprietary	2023/10
5	GPT-4o-mini-2024-07-18	1274	+4/-4	26088	OpenAI	Proprietary	2023/10
5	Claude 3.5 Sonnet	1270	+3/-3	56674	Anthropic	Proprietary	2024/4
5	Gemini-1.5-Flash-Exp-0827	1268	+5/-4	16780	Google	Proprietary	2023/11
5	Grok-2-Mini-08-13	1267	+4/-4	16731	xAI	Proprietary	2024/3
5	Meta-Llama-3.1-405b-Instruct	1266	+4/-4	27397	Meta	Llama 3.1 Community	2023/12

https://lmarena.ai/?leaderboard

Gemini (Google) | ChatGPT / GPT-4 (OpenAI) | Claude 3 (Anthropic) | Llama-3 (Meta) | Grok-2 (xAI)

Chiang et al. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. 2024.

Great Results With Transformers On NLP Tasks

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

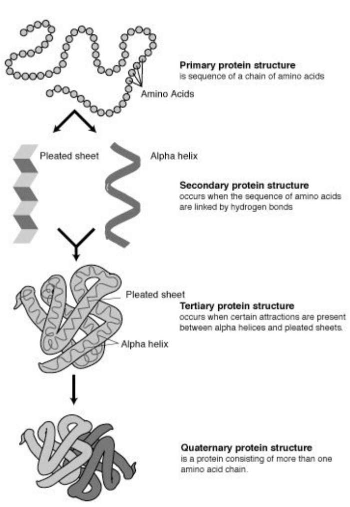
	Rank	Name	Model	URL	Score	BoolQ	CB	COPA	MultiRC	ReCoRD	RTE	WiC	WSC	AX-b	AX-g
+	1	Inspur Cloud	Hairuo		91.4	92.5	96.5/97.6	100.0	90.5/67.9	94.1/93.2	92.8	76.1	100.0	64.6	96.1/94.7
	2	JDExplore d-team	Vega v2		91.3	90.5	98.6/99.2	99.4	88.2/62.4	94.4/93.9	96.0	77.4	98.6	-0.4	100.0/50.0
+	3	Liam Fedus	ST-MoE-32B		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
	4	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
	5	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
	6	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
+	7	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
+	8	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

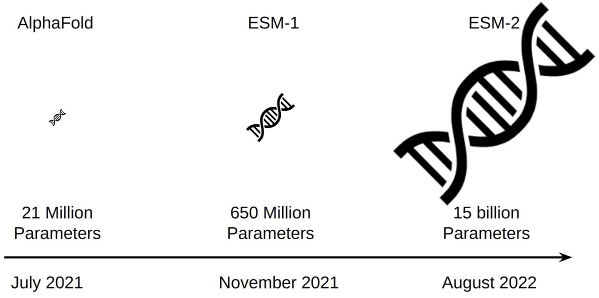
https://super.gluebenchmark.com/leaderboard

Wang et al. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. NeurIPS 2019.

Great Results With Transformers Outside NLP!

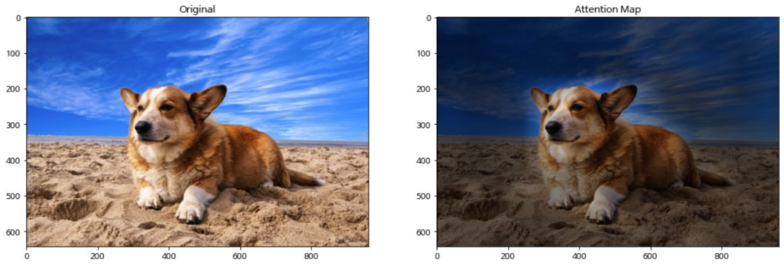
Protein Structure Prediction





Z. Lin et al. [Evolutionary-scale prediction of atomic-level protein structure with a language model](). Science 2023.

Image Classification



	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

Vision Transformer models pre-trained on JFT-300M dataset outperform ResNet-based baselines on all datasets, while taking substantially less compute to pretrain.

A. Dosovitskiy et al. [An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale](). ICLR 2021.

From Recurrence to Attention

Motivation for Transformer

- Minimize (or at least not increase) computational complexity per layer.
- Minimize path length between any pair of words to facilitate learning of long-range dependencies.
- Maximize the amount of computation that can be parallelized.

1. Transformer Motivation: Computational Complexity Per Layer

When sequence length (n) << representation dimension (d), the complexity per layer is lower for a Transformer model compared to RNN models.

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

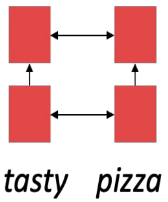
Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Table 1 of paper by A. Vaswani et al. Attention Is All You Need. NeurIPS 2017.

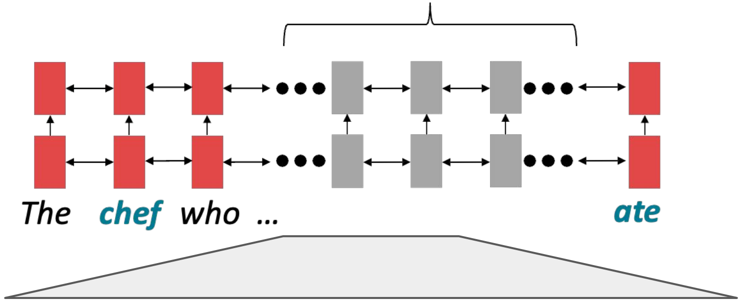
2. Transformer Motivation: Minimize Linear Interaction Distance

- RNN is unrolled “left-to-right”.
- It encodes linear locality: a useful heuristic!
- **Problem:** RNN takes **O(sequence length)** steps for distant word pairs to interact.
  - Hard to learn long-distance dependencies due to gradient problems.
  - Linear order of words is “baked in”; we already know sequential structure doesn't tell the whole story...

Nearby words often affect each other’s meanings



Info about ***chef*** has gone through O(sequence length) many layers!

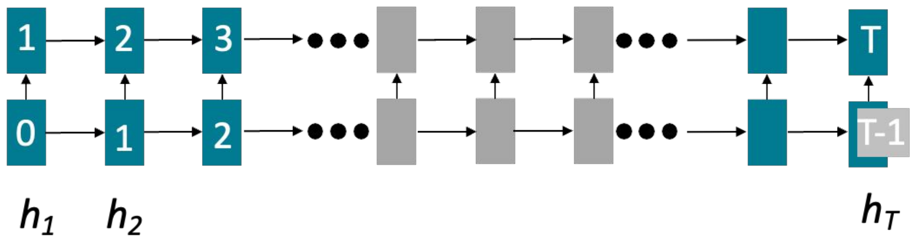


meticulously prepared the finest gourmet dishes, using only the freshest, most exquisite ingredients sourced from around the world, finally sat down at the end of a long day in the kitchen. Exhausted but satisfied, he carefully plated a small meal for himself, savoring each bite as he

3. Transformer Motivation: Maximize Parallelizability

Forward and backward passes have **O(sequence 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.

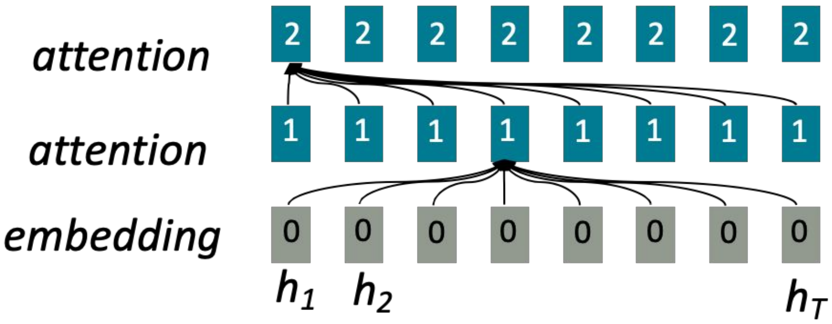


Numbers indicate min # of steps before a state can be computed

High-Level Architecture: Transformer is all about (Self) Attention

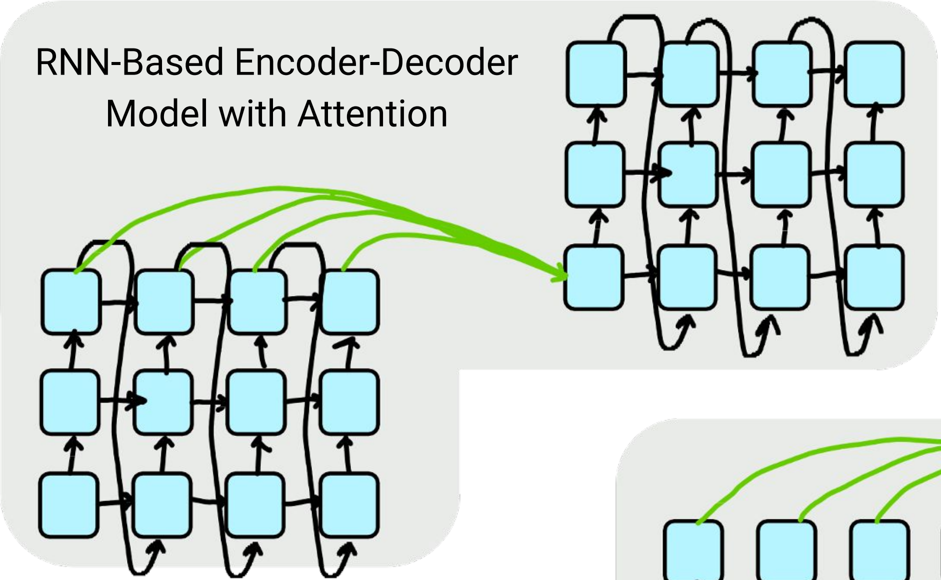
Earlier, we saw attention from the decoder to the encoder in a recurrent sequence-to-sequence model.

Self-attention is encoder-encoder (or decoder-decoder) attention where each word attends to each other word **within the input (or output)**.



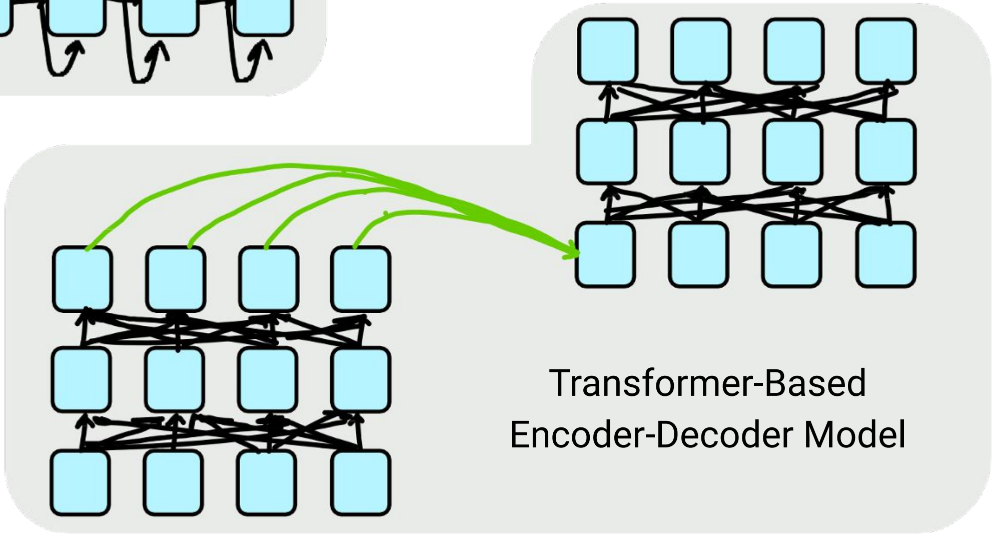
All words attend to all words in previous layer; most arrows here are omitted.

Computational Dependencies for Recurrence vs. Attention



Transformer Advantages:

- # unparallelizable operations does not increase with sequence length.
- Each word interacts with each other, so maximum interaction distance is O(1).



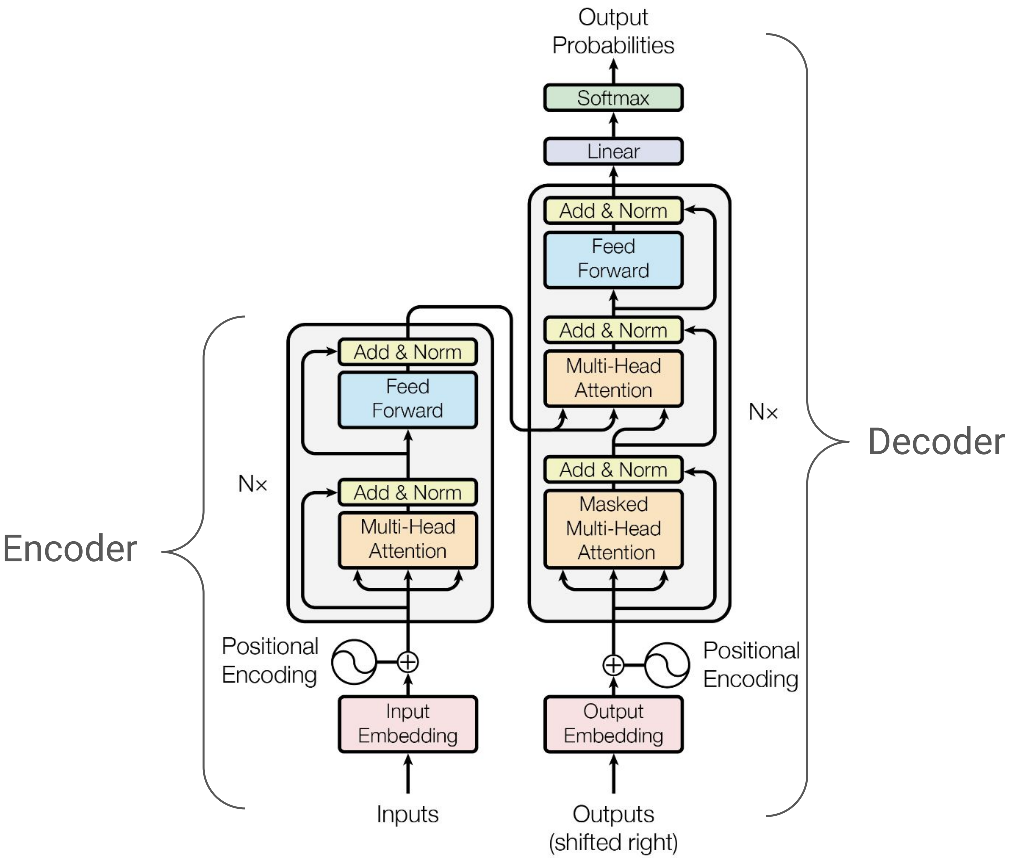
The Transformer Block

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

Next, we will learn exactly how the Transformer architecture works:

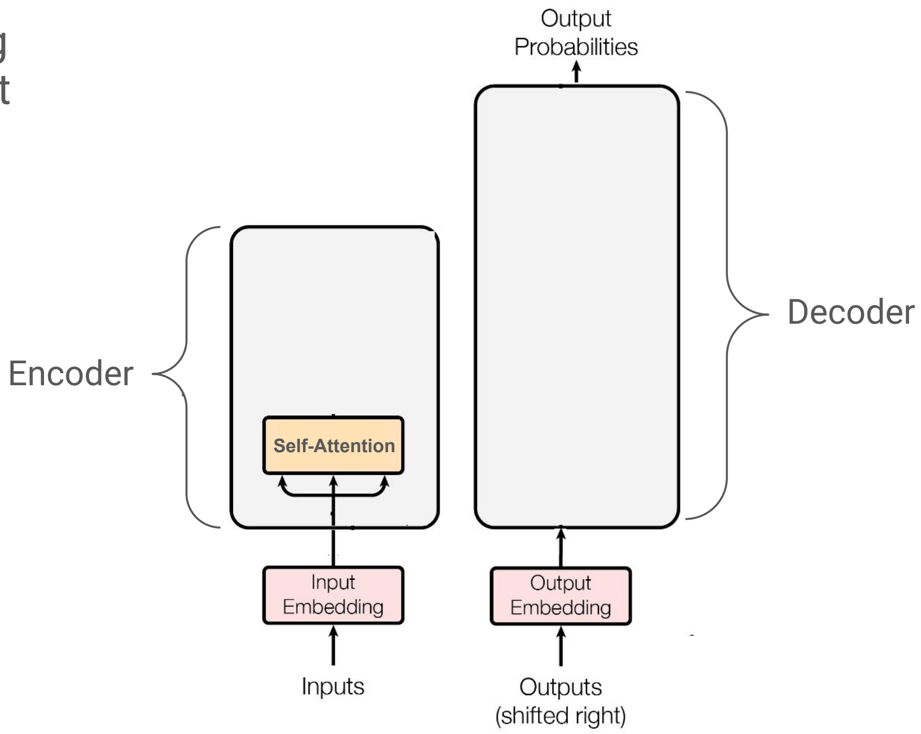
First, we will talk about the Encoder!

Next, we will go through the Decoder (which is quite similar)!



Encoder: Self-Attention

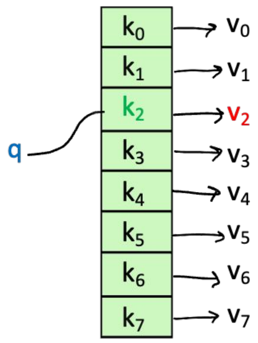
Self-Attention is the core building block of Transformer, so let's first focus on that!



Intuition for Attention Mechanism

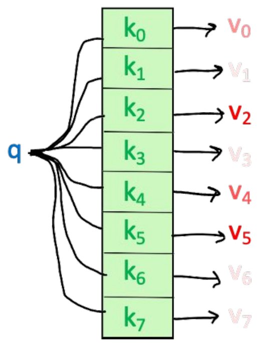
Let's think of attention as a "fuzzy" or approximate hashtable. To look up a value, we compare a query against keys in a table.

In a hashtable:



Each query (hash) maps to exactly one key-value pair.

In (self-)attention:



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.

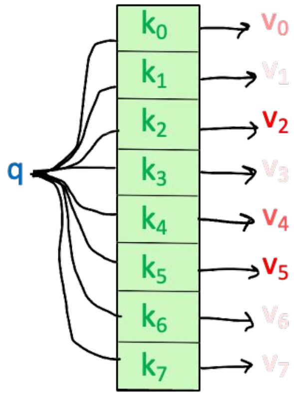
$$e_{ij} = q_i \cdot k_j$$

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

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

- Step 4: Take a weighted sum of values.

$$Output_i = \sum_j \alpha_{ij} v_j$$



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

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

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

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

$$E = QK^T$$

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

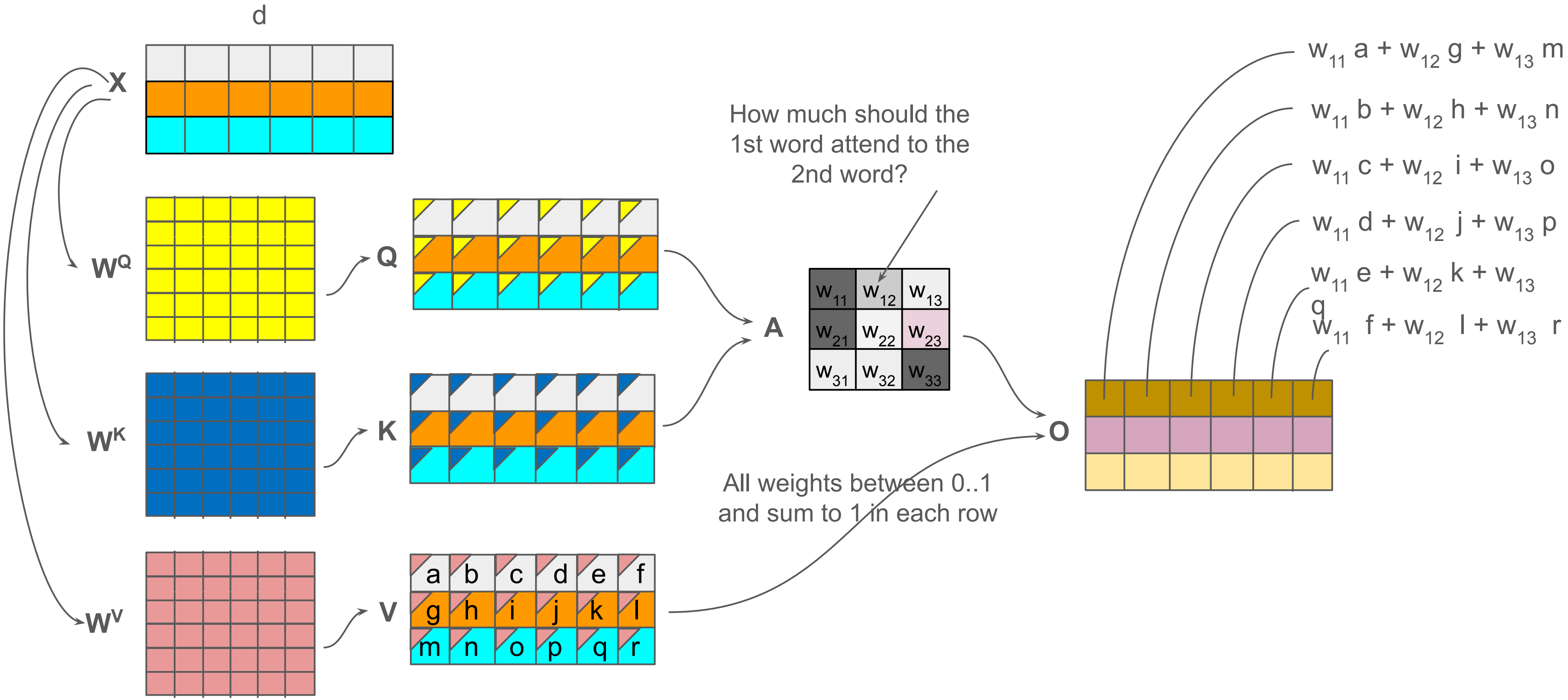
$$A = softmax(E)$$

- Step 4: Take a weighted sum of values.

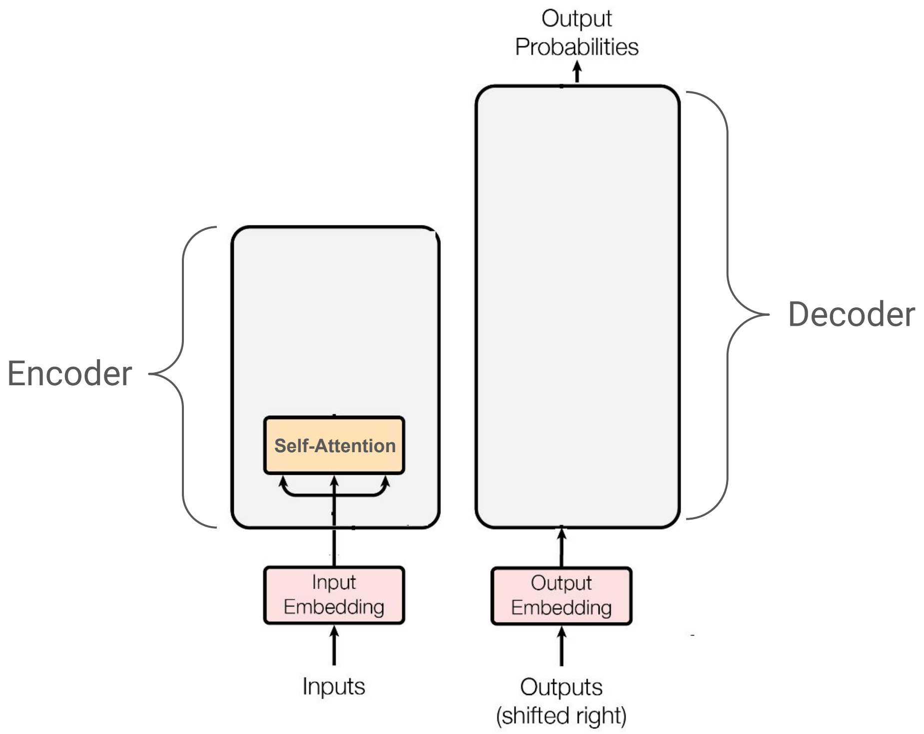
$$Output = AV$$

$$\boxed{Output = softmax\left(QK^T\right)V}$$

In Pictures (N = 3, d = 6, h = 1)



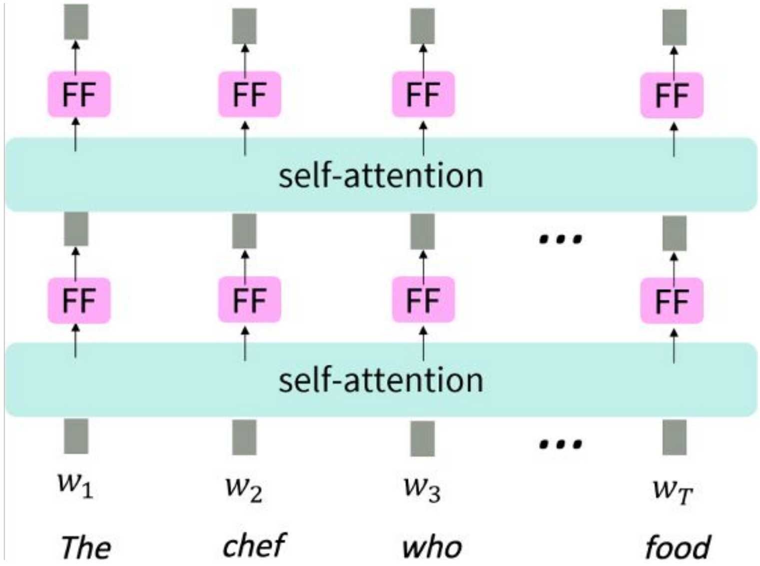
What We Have So Far: (Encoder) Self-Attention!

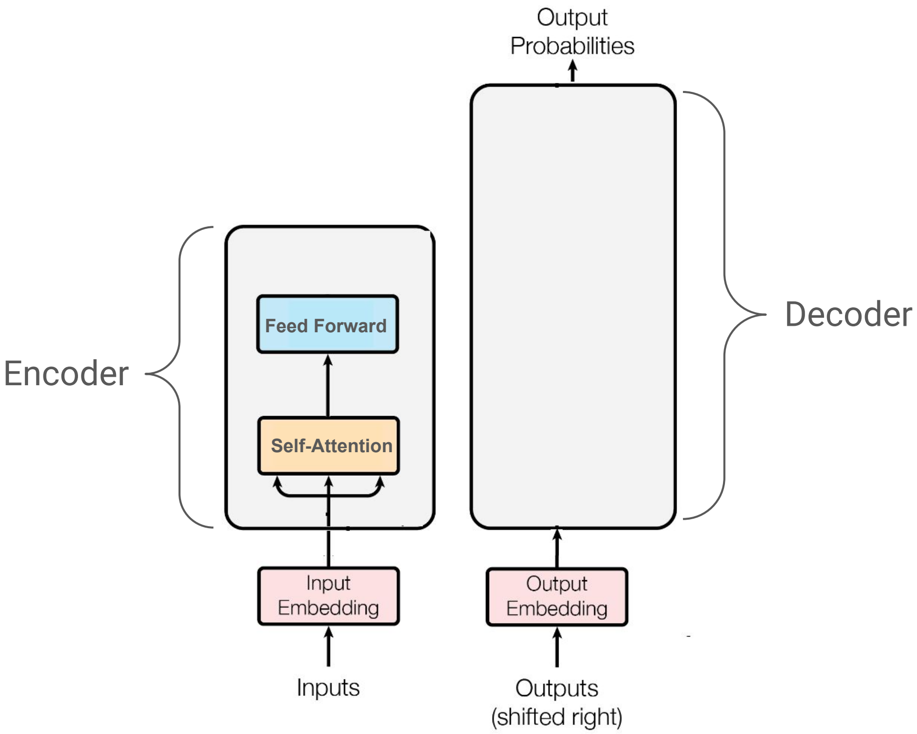


Attention Isn't Quite All You Need!

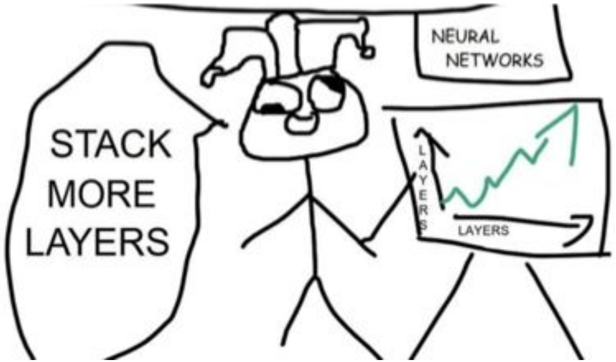
Equation for Feed Forward Layer

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

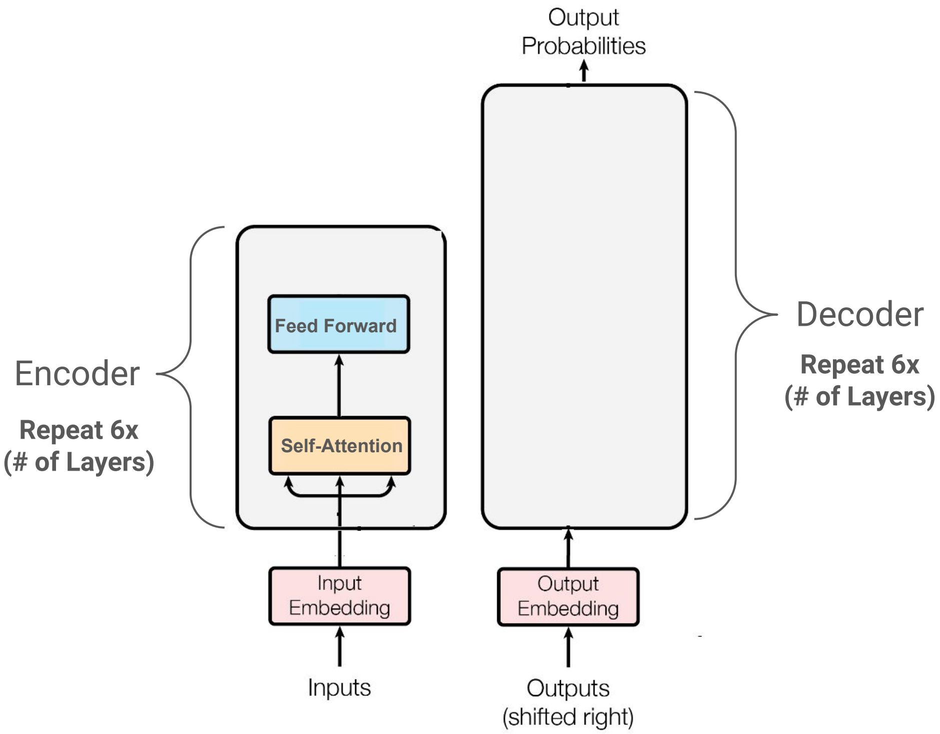




Making This Work For Deep Networks



Training Trick #1:Residual Connections
Training Trick #2: LayerNorm
Training Trick #3: Scaled Dot Product Attention



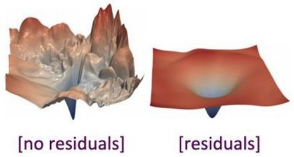
Training Trick #1: Residual Connections [He et al., 2016]

- 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!

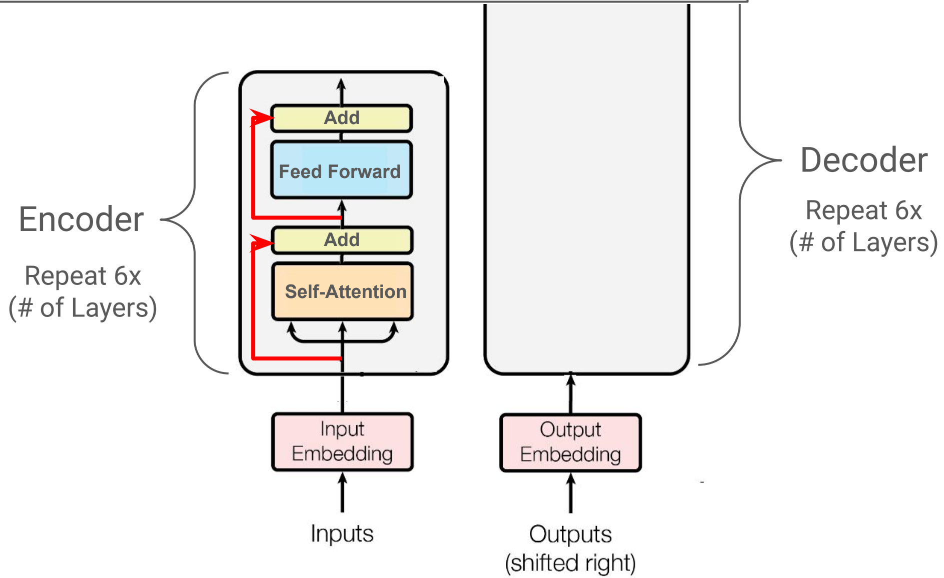
$$x_\ell = F(x_{\ell-1}) + x_{\ell-1}$$

- This prevents the network from "forgetting" or distorting important information as it is processed by many layers.

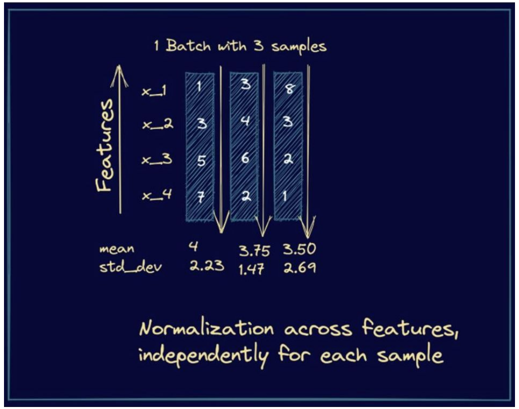
Residual connections are also thought to smooth the loss landscape and make training easier!



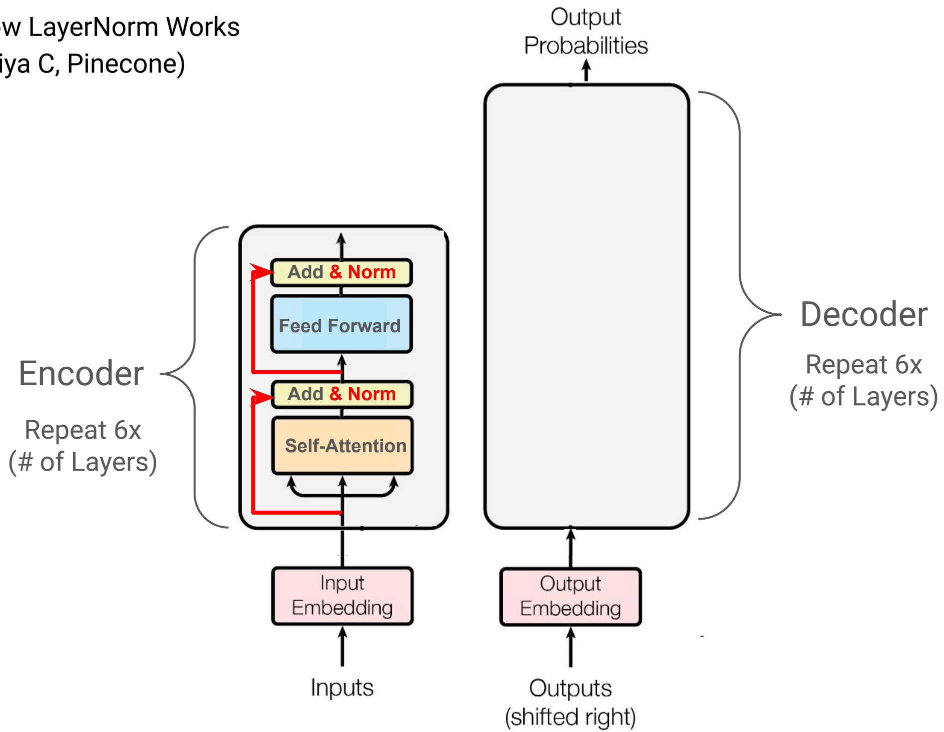
[Loss landscape visualization, Li et al., 2018, on a ResNet]



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



An Example of How LayerNorm Works
(Image by Bala Priya C, Pinecone)



Mean:

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$$

Standard Deviation:

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

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

Training Trick #3: Scaled Dot Product Attention

After LayerNorm, the mean and variance of vector elements is 0 and 1, respectively.

However, the dot product still tends to take on extreme values, as its variance scales with dimensionality d_k.

Quick Statistics Review:

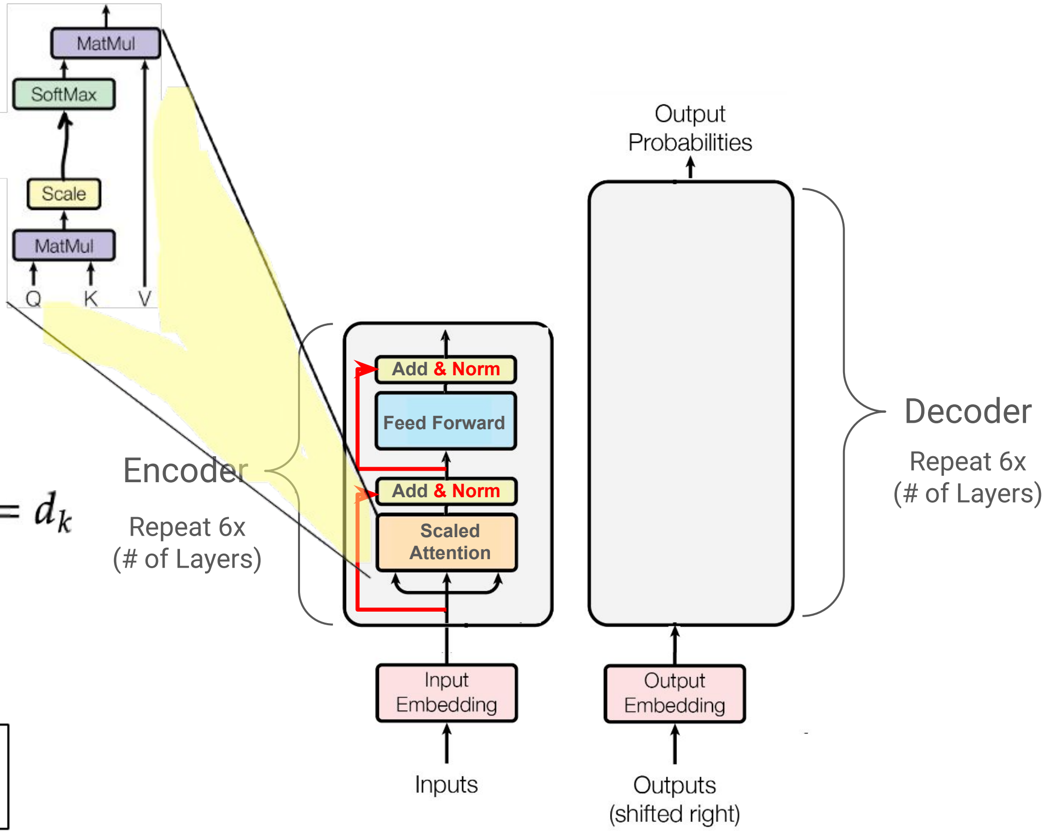
Mean of sum = sum of means = $d_k * 0 = 0$

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:

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



Positional Encodings

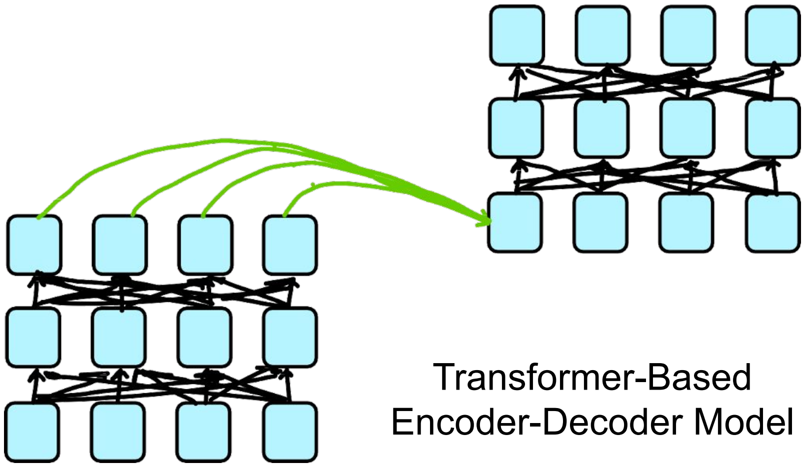
$$Output = softmax\left(QK^T / \sqrt{d_k}\right)V$$

We're almost done with the Encoder, but we have a problem!

Consider this sentence: "Man eats small dinosaur."

Order doesn't impact the network at all!

This seems wrong given that word order does have meaning in many languages, including English!



Transformer-Based
Encoder-Decoder Model

Positional Encodings

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 $i \in \{1,2, \dots, T\}$ (called position vector).

Don't worry about what the p_i are made of yet!

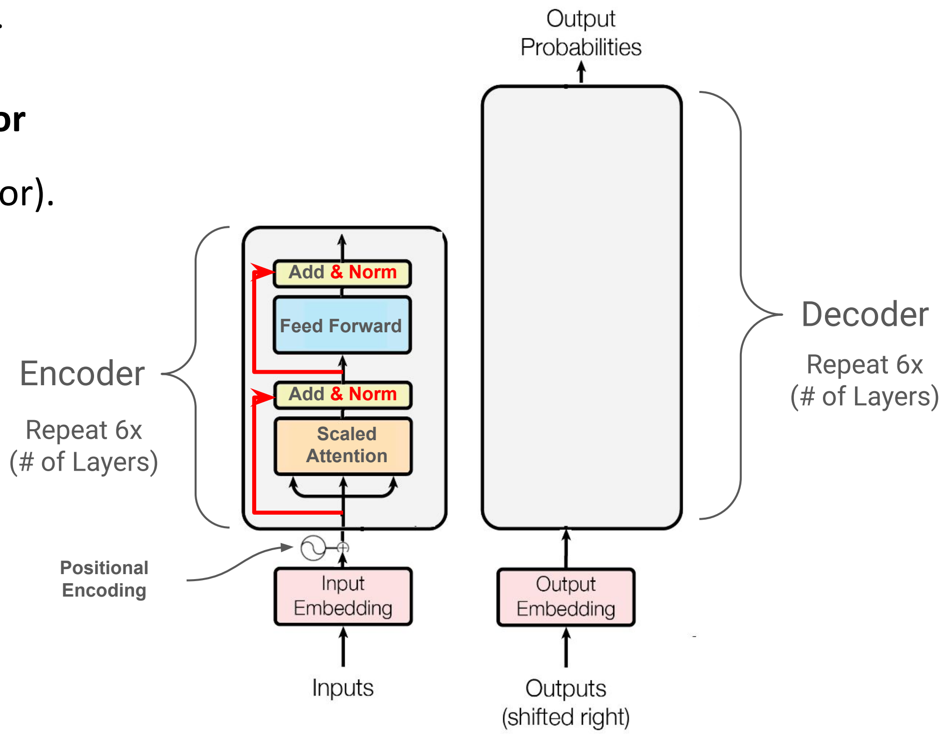
Easy to incorporate this info into self-attention block:

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.

Then:

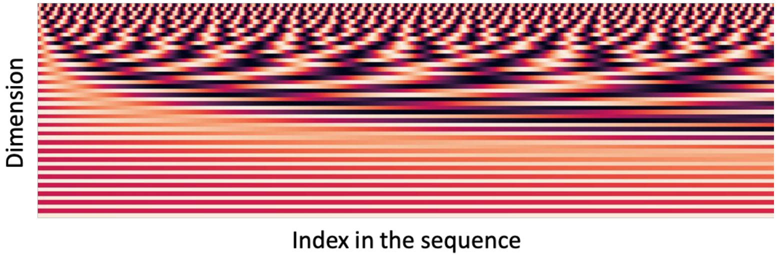
$$v_i = \tilde{v}_i + p_i$$
$$q_i = \tilde{q}_i + p_i$$
$$k_i = \tilde{k}_i + p_i$$



Position Representation Vectors Using Sinusoids (Original)

Sinusoidal position representations: concatenate sinusoidal functions of varying periods

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

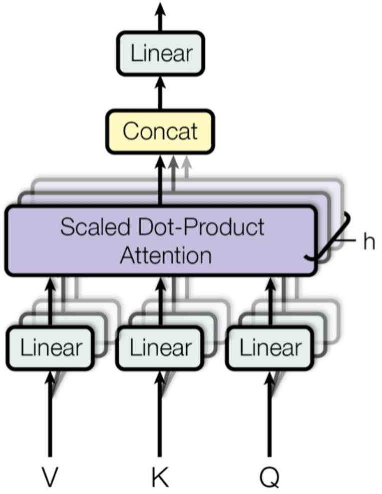


We will study pros and cons when we look at two alternatives of such **absolute** position encodings: **relative** position encodings and **rotary** position encodings.

Image: https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/

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

High-Level Idea: Perform self-attention multiple times in parallel and combine the results.



[Vaswani et al. 2017]

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 $x_i^\top Q^\top K x_j$ is high, but maybe we want to focus on different j for different reasons?

Define **multiple attention "heads"** through multiple Q, K, V matrices!

Let $Q_P, K_P, V_P \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and P ranges from 1 to h.

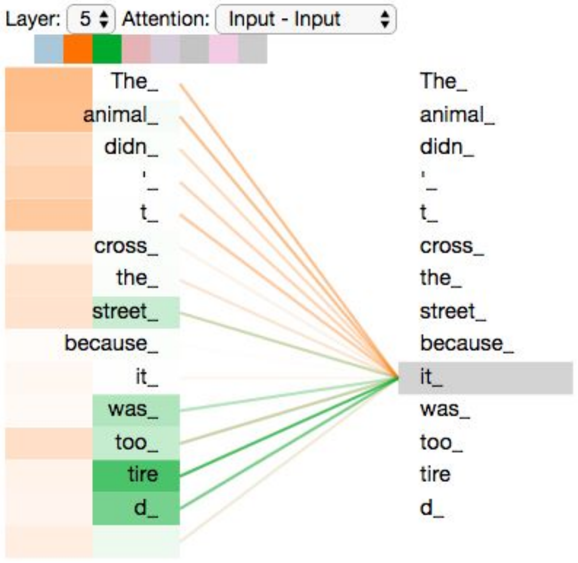
Each attention head performs attention independently:

$$\text{output}_P = \text{softmax}(XQ_P K_P^\top X^\top) * XV_P, \text{where output}_P \in \mathbb{R}^{d/h}$$

Then the outputs of all the heads are combined!

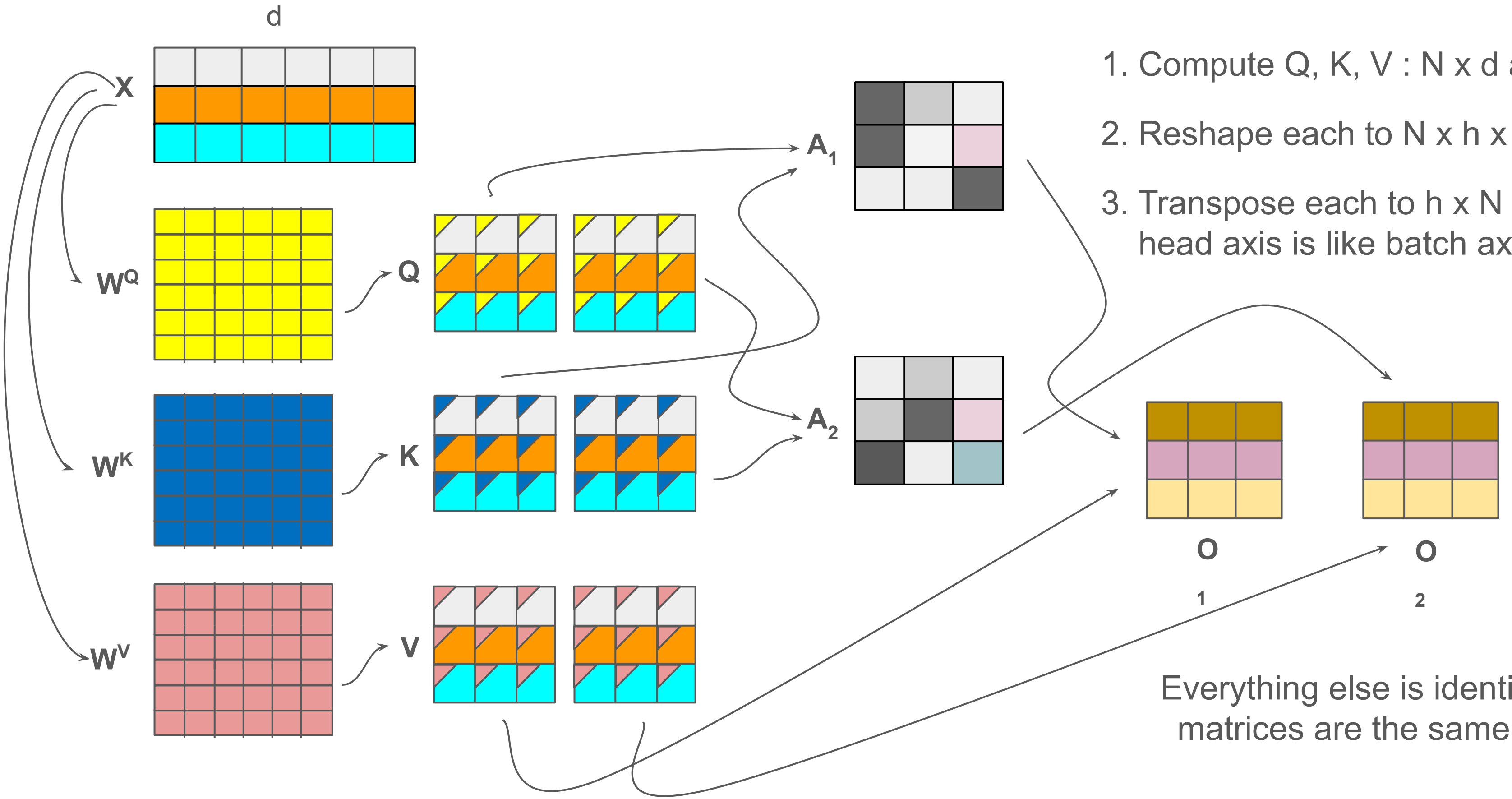
$$\text{output} = Y[\text{output}_1; \ldots; \text{output}_h], \text{where } Y \in \mathbb{R}^{d \times d}$$

Each head gets to "look" at different things, and construct value vectors differently.



In encoding the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired". The model's representation of the word "it" thus bakes in some of the representation of both "animal" and "tired".
https://jalammar.github.io/illustrated-transformer/

In Pictures (N = 3, d = 6, h = 2)



1. Compute Q, K, V : N x d as before
2. Reshape each to N x h x d/h
3. Transpose each to h x N x d/h; now head axis is like batch axis

Everything else is identical and matrices are the same sizes!

The Story So Far ...

We've completed the Encoder!

Next lecture we will look at the Decoder...