

# Self-Attention and Transformers

Deep Natural Language Processing Class, 2022  
Paweł Budzianowski

English quality

Be more verbal

# Scores

# Group project

# Discovering building blocks

1. Meaning
2. Word Vectors
3. New NLP task - language modelling
4. New family of ML models - recurrent neural networks
5. Seq2Seq
6. Attention
7. LSTMs, Bidirectional RNNs and Multilayer RNNs
8. Self-Attention (today)
9. Transformers (today)

---

# 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

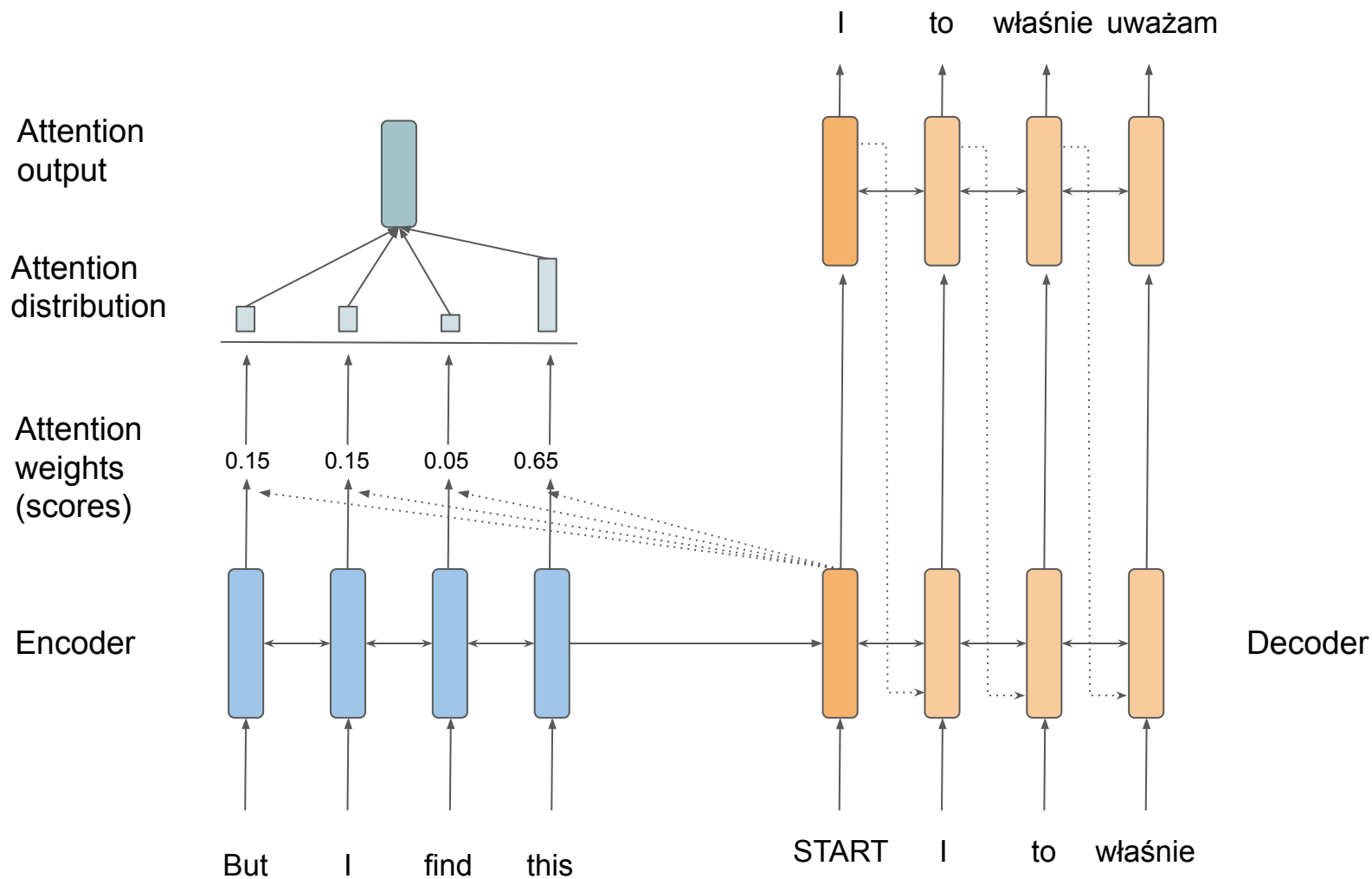
**Illia Polosukhin\* ‡**

illia.polosukhin@gmail.com

# RNN as default thing to do

- The classic strategy for NLP was to encode the sentence with a bidirectional LSTM
- Define your output as a sequence and use an LSTM to generate it.
- Use attention to look holistically at the context.





# Making modelling better

No new ways of looking at problems - we stay with Seq2Seq paradigm.

But rather new way of modelling existing ones.

Issues with recurrent models: Linear interaction distance

# Issues with recurrent models: Linear interaction distance

- RNNs are unrolled “left-to-right”
- This encodes linear locality: a useful heuristic
  - Nearby words often affect each other’s meanings

# Issues with recurrent models: Linear interaction distance

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

LM task - “O, w jednym zdaniu może się dużo zmieścić. Może się wszystko zmieścić. Może się całe życie zmieścić. Zdanie jest wymiarem świata, powiedział filozof. Tak, on. Czasem sobie myślę, czy nie dlatego tyle słów musimy przez życie powiedzieć, aby się mogło z nich wytopić to jedno \_\_\_\_\_” (Wiesław Myśliwski, Traktat o łuskaniu fasoli)

# Issues with recurrent models: Linear interaction distance

**$O(\text{sequence length})$**  steps for distant word pairs to interact means:

- Hard to learn long-distance dependencies (because gradient passing)
- Linear order of words is incorporated into the model, we already know linear order isn't the right way to think about sentences.

# Issues with recurrent models: Lack of parallelizability

Forward and backward passes have  $O(T)$  non parallelizable operations

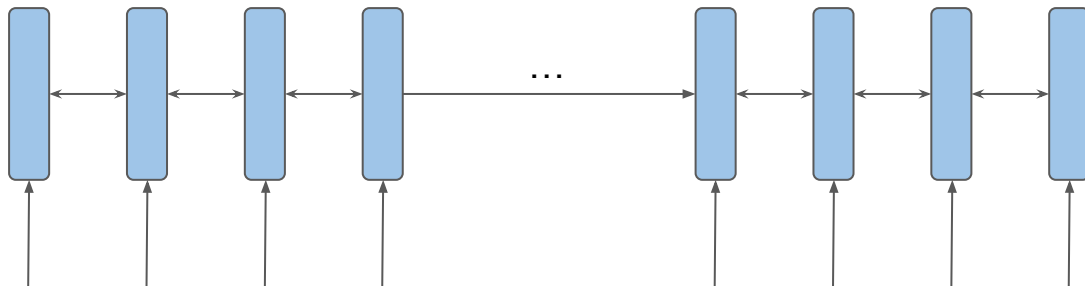
- GPUs can perform a bunch of independent computation at once
- But future RNN hidden states can't be computed in full before past RNN hidden states have been computed
- Inhibits training on very large datasets

We can't parallelize over the time dimension.

# Issues with recurrent models: Lack of parallelizability

Forward and backward passes have  $O(T)$  non parallelizable operations

- GPUs can perform a bunch of independent computation 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



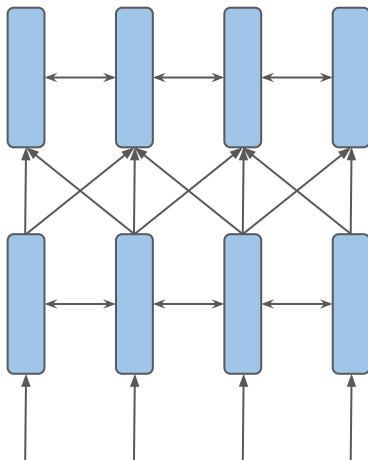


If not recurrence, then what? Word windows!

# If not recurrence, then what? Word windows!

Word window models aggregate local contexts

- Also known as 1D convolution!
- Number of non-parallelizable operations does not increase with sequence length



# If not recurrence, then what? Word windows!

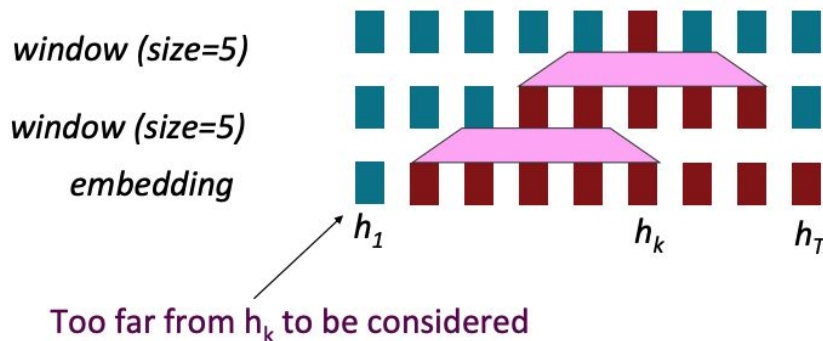
Word window models aggregate local contexts

- What about long-distance dependencies?
  - Stacking word window layers allows interaction between farther words
- Maximum interaction distance = sequence length / window size
  - But if your sequences are too long, you will just ignore long-distance context

# If not recurrence, then what? Word windows!

Word window models aggregate local contexts

- What about long-distance dependencies?
  - Stacking word window layers allows interaction between farther words
- Maximum interaction distance = sequence length / window size
  - But if your sequences are too long, you will just ignore long-distance context



# If not recurrence, then what? Attention!

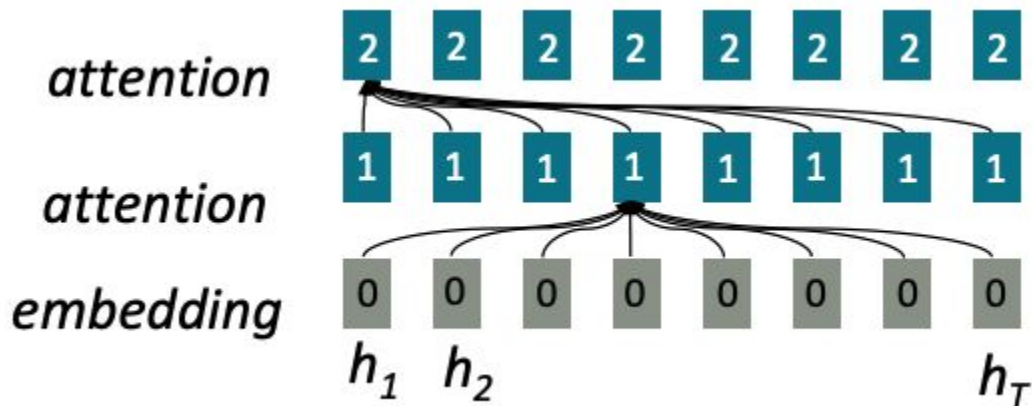
**Attention** treats each word's representation as a **query** to access and incorporate information from a **set of values**.

- We saw attention from the **decoder** to the **encoder**
- Let's look at attention within a **single sentence**

# If not recurrence, then what? Attention!

**Attention** treats each word's representation as a **query** to access and incorporate information from a **set of values**.

- We saw attention from the **decoder** to the **encoder**
- Let's look at attention within a **single sentence**



# If not recurrence, then what? Attention!

**Attention** treats each word's representation as a **query** to access and incorporate information from a **set of values**.

- We saw attention from the **decoder** to the **encoder**
- Let's look at attention within a **single sentence**

Number of non-parallelizable operations does not increase with sequence length.

Maximum interaction distance:  $O(1)$  since all words interact at every layer.

# Self-Attention

Attention operates on **queries**, **keys**, and **values**

- We have some **queries**  $q_1, q_2, \dots, q_T$  Each query is  $q_i \in \mathbb{R}^d$
- We have some **keys**,  $k_1, k_2, \dots, k_T$  Each key is  $k_i \in \mathbb{R}^d$
- We have some **values**  $v_1, v_2, \dots, v_T$  Each value is  $v_i \in \mathbb{R}^d$



# Self-Attention

Attention operates on **queries**, **keys**, and **values**

- We have some **queries**  $q_1, q_2, \dots, q_T$  Each query is  $q_i \in \mathbb{R}^d$
- We have some **keys**,  $k_1, k_2, \dots, k_T$  Each key is  $k_i \in \mathbb{R}^d$
- We have some **values**  $v_1, v_2, \dots, v_T$  Each value is  $v_i \in \mathbb{R}^d$

In **self-attention**, all three above come from the same source!

- We can let  $v_i = k_i = q_i = x_i$

# Self-Attention

Attention operates on **queries**, **keys**, and **values**

- We have some **queries**  $q_1, q_2, \dots, q_T$  Each query is  $q_i \in \mathbb{R}^d$
- We have some **keys**,  $k_1, k_2, \dots, k_T$  Each key is  $k_i \in \mathbb{R}^d$
- We have some **values**  $v_1, v_2, \dots, v_T$  Each value is  $v_i \in \mathbb{R}^d$

In **self-attention**, all three above come from the same source!

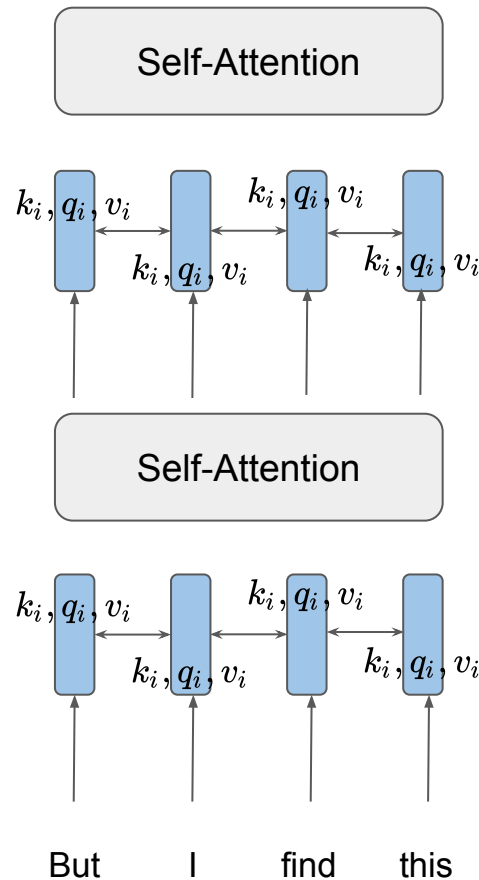
- We can let  $v_i = k_i = q_i = x_i$
- The dot product self-attention operation is as follows:

$$e_{ij} = q_i^T k_j \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_l \exp(e_{il})} \qquad \text{output}_i = \sum_j \alpha_{ij} v_j$$

# Self-attention as a building block

In the diagram at the right, we have stacked self-attention blocks, like we might stack LSTM layers

Q: Can self-attention a drop-in replacement for recurrence?

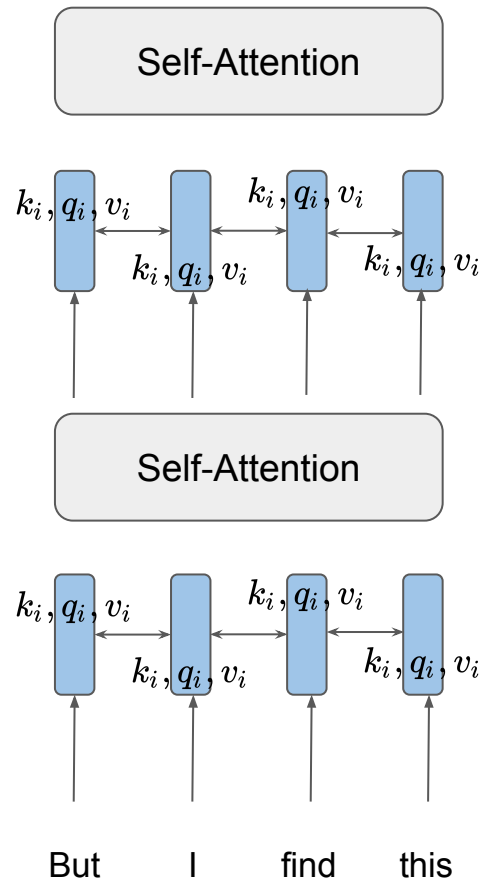


# Self-attention as a building block

In the diagram at the right, we have stacked self-attention blocks, like we might stack LSTM layers

Q: Can self-attention a drop-in replacement for recurrence?

A: No, it has a few issues.

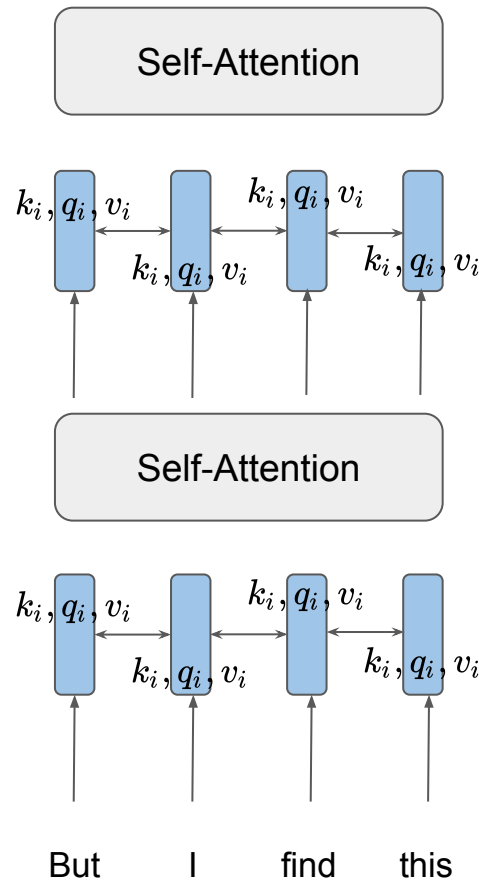


# Self-attention as a building block

In the diagram at the right, we have stacked self-attention blocks, like we might stack LSTM layers

Q: Can self-attention a drop-in replacement for recurrence?

A: No, it has a few issues.



# Barriers and solutions for Self-Attention as building blocks

1. First, self-attentions are operations on sets.

It doesn't know what is the order.

# 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.

# 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**:

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

Easy to incorporate into our self-attention block just add the  $p$  to our inputs.



# Sequence order

## Requirements:

- It should output a unique encoding for each time-step (word's position in a sentence)
- Distance between any two time-steps should be consistent across sentences with different lengths.
- Our model should generalize to longer sentences without any efforts. Its values should be bounded.
- It must be deterministic.

# Position representation vectors through sinusoids

Sinusoidal position representation: concatenate sinusoidal functions of varying periods

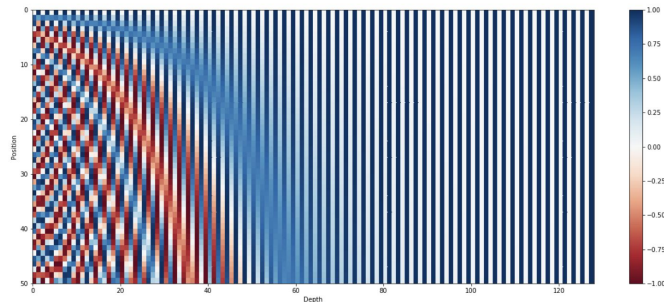
$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \dots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$

Features:

- Periodicity indicates that maybe absolute position isn't as important
- Maybe can extrapolate to longer sequences as periods restart

Issue:

- Not learnable, the extrapolation doesn't really work



# Position representation vectors learned from scratch

Learned absolute position representations: let all  $p_i$  be learnable parameters.

Pros:

- Flexibility each position gets to be learned to fit the data

Cons:

- Definitely can't extrapolate to indices outside  $1, \dots, T$

It's being a standard default thing to do.

Some works try more flexible representations of positions [Shaw et al., 2018]

# Barriers and solutions for Self-Attention as building blocks

1. First, self-attentions are operations on sets.

It doesn't know what is the order.

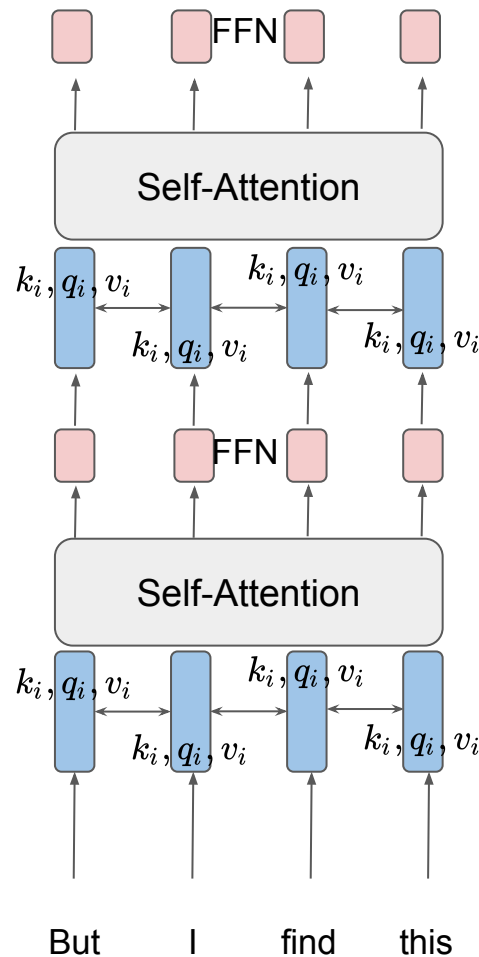
2. No deep-learning to model nonlinearities!

# Adding nonlinearities in self-attention

Note that all we do is re-averaging **value** vectors in stacked self-attentions.

Add a **feed-forward network** to post-process each output vector.

$$W_2(\text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$



# Barriers and solutions for Self-Attention as building blocks

1. First, self-attentions are operations on sets. It doesn't know what is the order.
2. No deep-learning to model nonlinearities!
3. Need to ensure we don't look at the future when predicting a sequence

# Masking the future in self-attention

- To use self-attention in **decoders**, we need to ensure we can't look into the future.
- At every timestep, we could change the set of **keys and queries** to include only past words (inefficient!)

# Masking the future in self-attention

- To use self-attention in **decoders**, we need to ensure we can't look into 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  $e_{ij} = -\infty$

	STA RT	I	find	this
STA RT				
I				
find				
this				



# Barriers and solutions for Self-Attention as building blocks

1. First, self-attentions are operations on sets. It doesn't know what is the order.
2. No deep-learning to model nonlinearities!
3. Need to ensure we don't look at the future when predicting a sequence

# Necessities for a self-attention building block:

- Self-attention:
  - The basis of the method

# Necessities for a self-attention building block:

- Self-attention:
  - The basis of the method
- Position representations:
  - Specify the sequence order, since self-attention is an unordered function of its input
- Nonlinearities:
  - At the output of the self-attention block
  - A simple feed-forward network

# Necessities for a self-attention building block:

- Self-attention:
  - The basis of the method
- Position representations:
  - Specify the sequence order, since self-attention is an unordered function of its input
- Nonlinearities:
  - At the output of the self-attention block
  - A simple feed-forward network
- Masking:
  - In order to parallelize operations while not looking at the future
  - Keeps information about the future from “leaking”

# Necessities for a self-attention building block:

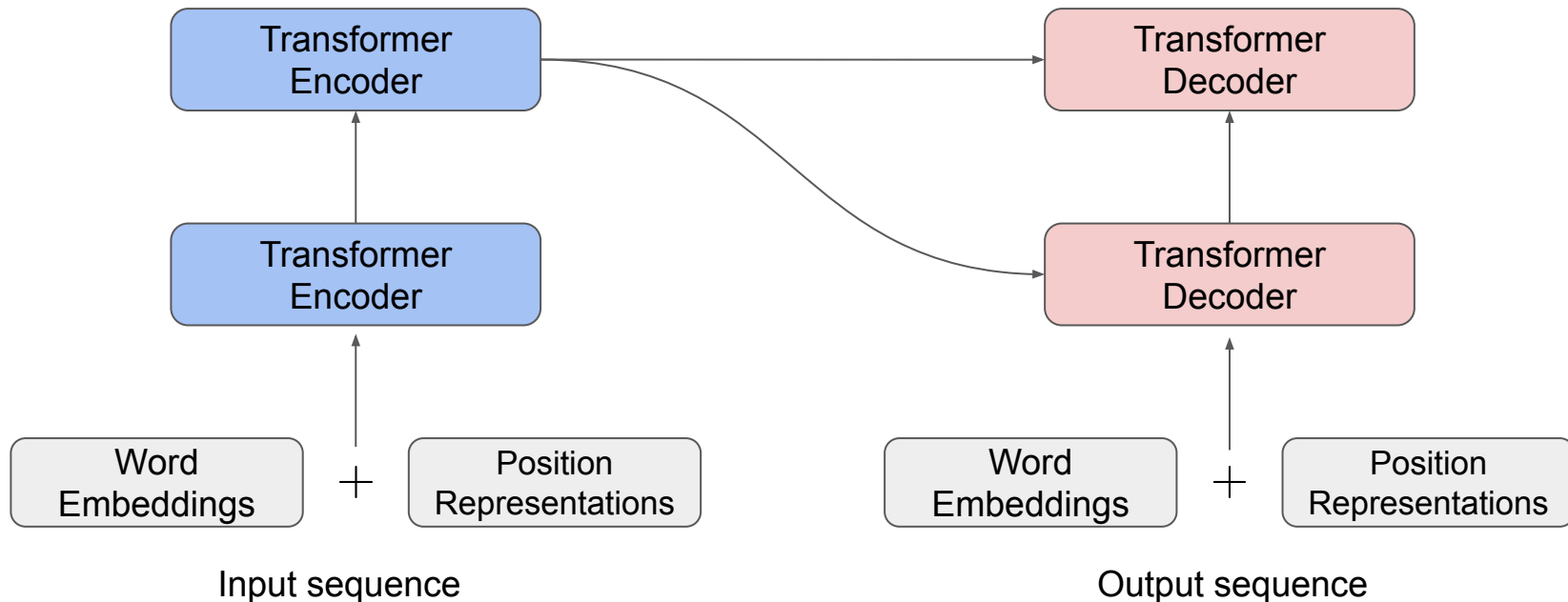
- Self-attention:
  - The basis of the method
- Position representations:
  - Specify the sequence order, since self-attention is an unordered function of its input
- Nonlinearities:
  - At the output of the self-attention block
  - A simple feed-forward network
- Masking:
  - In order to parallelize operations while not looking at the future
  - Keeps information about the future from “leaking”

These are the building blocks of the **Transformer**!



# Attention is all you need [Vasvani et al. 2017]

Let's look at the Transformer Encoder and Decoder Blocks at a high level.



# Transformer Encoder

What's left in a Transformer Encoder Block that we haven't covered?

1. Key-query-value parameters: how do we get the  $k, q, v$  vectors from a single word embedding?



# Transformer Encoder

What's left in a Transformer Encoder Block that we haven't covered?

1. Key-query-value parameters: how do we get the  $k, q, v$  vectors from a single word embedding?
2. Multi-headed attention: Attend to multiple places in a single layer.

# Transformer Encoder

What's left in a Transformer Encoder Block that we haven't covered?

1. Key-query-value parameters: how do we get the  $k, q, v$  vectors from a single word embedding?
2. Multi-headed attention: Attend to multiple places in a single layer.
3. Tricks to help with training
  - a. Residual connections
  - b. Layer normalization
  - c. Scaling the dot product
  - d. These tricks don't improve what the model is able to do; they help to improve the training process.

A big lesson here (quite sad though)!

# Key-query-value Attention

- We saw that self-attention is when keys, queries, and values come from the same source. The Transformer does this in a particular **way**:
- Let  $x_1, \dots, x_T$  be input vectors to the encoder,  $x_i \in \mathbb{R}^d$
- Then keys, queries, and values are:
  - $k_i = Kx_i$ , where  $K \in \mathbb{R}^{d \times d}$  is the key matrix
  - $q_i = Qx_i$ , where  $Q \in \mathbb{R}^{d \times d}$  is the query matrix
  - $v_i = Vx_i$ , where  $V \in \mathbb{R}^{d \times d}$  is the value matrix
- These matrices allow different aspects of the  $x$  vectors to be used/emphasized in each of the three roles.

# Efficiency, you fool!

- Let's look at how key-query-value attention is computed, in matrices.
  - Let  $X = [x_1; \dots; x_t] \in \mathbb{R}^{T \times d}$  be the concatenation of input vectors.
  - First, note that  $XK \in \mathbb{R}^{T \times d}$ ,  $XQ \in \mathbb{R}^{T \times d}$ ,  $XV \in \mathbb{R}^{T \times d}$
  - The output is defined as  $\text{output} = \text{softmax}(XQ(XK)^T) \times XV$

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

$$\begin{array}{c} \begin{array}{ccc} \boxed{XQ} & & \\ & \boxed{K^T X^T} & \\ & & \boxed{XQK^T X^T} \end{array} & = & \\ & & \in \mathbb{R}^{T \times T} \\ \text{softmax} \begin{array}{ccc} \boxed{XQK^T X^T} & \boxed{XV} & \\ & = & \\ & \boxed{\text{output}} & \\ & & \in \mathbb{R}^{T \times d} \end{array} \end{array}$$

# Efficiency, you fool!

- Let's look at how key-query-value attention is computed, in matrices.
  - Let  $X = [x_1; \dots; x_t] \in \mathbb{R}^{T \times d}$  be the concatenation of input vectors.
  - First, note that  $XK \in \mathbb{R}^{T \times d}$ ,  $XQ \in \mathbb{R}^{T \times d}$ ,  $XV \in \mathbb{R}^{T \times d}$
  - The output is defined as  $\text{output} = \text{softmax}(XQ(XK)^T) \times XV$

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

No sequential calls, no for loops!

$$\begin{array}{c} \begin{array}{ccc} \boxed{XQ} & & \\ & \boxed{K^T X^T} & \\ & & \end{array} = \boxed{XQK^T X^T} \\ & & \in \mathbb{R}^{T \times T} \\ \\ \text{softmax} \left( \boxed{XQK^T X^T} \right) \boxed{XV} = \boxed{\text{output}} \\ & & \in \mathbb{R}^{T \times d} \end{array}$$

# Multi-headed attention

- What if we want to look in multiple places in the sentence at once?
  - For word  $i$ , self-attention “looks” where  $x_i^T Q^T K x_j$  is high, but maybe we want to focus on different  $j$  for different reasons

# Multi-headed attention

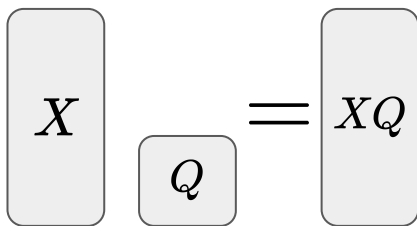
- What if we want to look in multiple places in the sentence at once?
  - For word  $i$ , self-attention “looks” where  $x_i^T Q^T K x_j$  is high, but maybe we want to focus on different  $j$  for different reasons
- We will define **multiple attention heads** through multiple  $Q, K, V$  matrices.
- Let  $Q_l, K_l, V_l \in \mathbb{R}^{d \times \frac{d}{h}}$  where  $h$  is the number of attention heads, and  $l$  ranges from 1 to  $h$ .

# Multi-headed attention

- What if we want to look in multiple places in the sentence at once?
  - For word  $i$ , self-attention “looks” where  $x_i^T Q^T K x_j$  is high, but maybe we want to focus on different  $j$  for different reasons
- We will define **multiple attention heads** through multiple  $Q, K, V$  matrices.
- Let  $Q_l, K_l, V_l \in \mathbb{R}^{d \times \frac{d}{h}}$  where  $h$  is the number of attention heads, and  $l$  ranges from 1 to  $h$ .
- Each attention head performs attention independently.
  - $\text{output}_l = \text{softmax}(X Q_l K_l^T X^T) X V_l$  where output
- Each attention performs attention independently:
- Outputs of all the heads are combined now:
  - $\text{output} = Y[\text{output}_1; \dots; \text{output}_h]$ , where  $Y \in \mathbb{R}^{d \times d}$

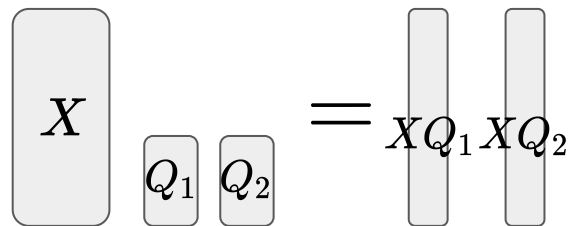


# Multi-headed attention



A diagram illustrating a single-head attention operation. On the left, a tall, light-gray rounded rectangle contains the letter  $X$ . To its right is a smaller, light-gray rounded rectangle containing the letter  $Q$ . An equals sign is placed between these two rectangles and a third, tall, light-gray rounded rectangle on the right, which contains the expression  $XQ$ .

$$X \quad Q = XQ$$



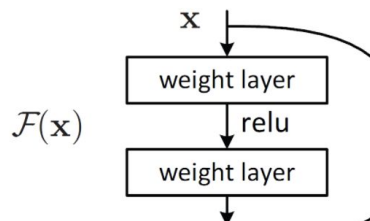
A diagram illustrating a multi-head attention operation. On the left, a tall, light-gray rounded rectangle contains the letter  $X$ . To its right are two smaller, light-gray rounded rectangles containing the labels  $Q_1$  and  $Q_2$ . An equals sign is placed between these rectangles and two tall, narrow, light-gray rounded rectangles on the right. The first narrow rectangle contains the expression  $XQ_1$  and the second contains the expression  $XQ_2$ .

$$X \quad Q_1 \quad Q_2 = XQ_1 \quad XQ_2$$

# Training tricks

# Residual connections

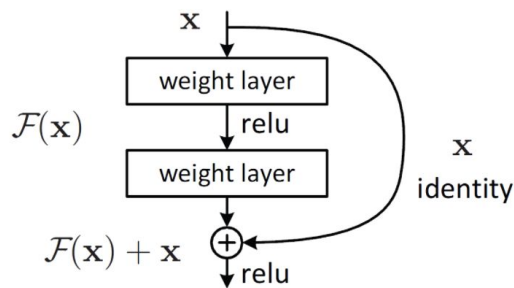
- Residual connections are a trick to help models train better.



Residual connections are thought to make the loss landscape considerably smoother.

# Residual connections

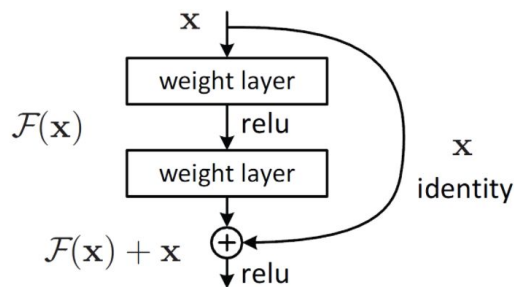
- Residual connections are a trick to help models train better.



Residual connections are thought to make the loss landscape considerably smoother.

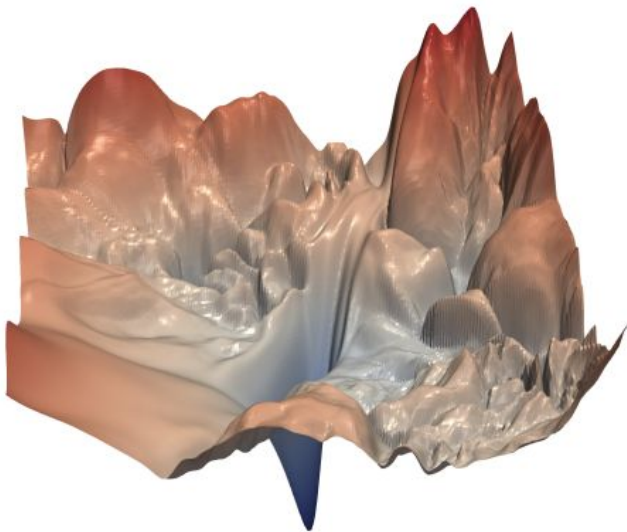
# Residual connections

- Residual connections are a trick to help models train better.

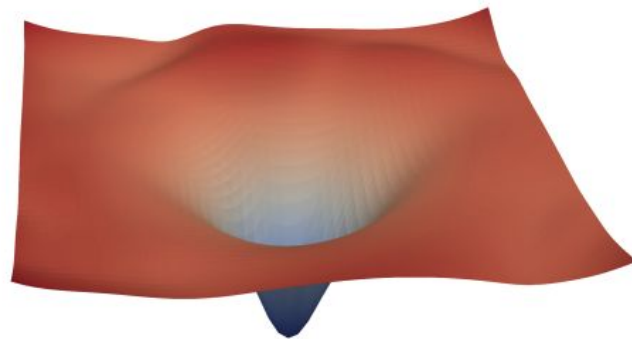


Residual connections are thought to make the loss landscape considerably smoother.

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial H} \frac{\partial H}{\partial x} = \frac{\partial L}{\partial H} \left( \frac{\partial F}{\partial x} + 1 \right) = \frac{\partial L}{\partial H} \frac{\partial F}{\partial x} + \frac{\partial L}{\partial H}$$



(a) without skip connections



(b) with skip connections

# Layer normalization [Ba et al., 2016]

- **Layer normalization** is a trick to help models train faster
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation within **each layer**

# Layer normalization [Ba et al., 2016]

- **Layer normalization** is a trick to help models train faster
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation within **each layer**
- Let  $x \in \mathbb{R}^d$  be an individual vector in the model
- Let  $\mu = \sum_{j=1}^d x_j$  , this the mean
- Let  $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^d (x_j - \mu)^2}$  , this is the standard deviation
- Let  $\gamma, \beta \in \mathbb{R}^d$  be learned gain and bias parameters



# Layer normalization [Ba et al., 2016]

- **Layer normalization** is a trick to help models train faster
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation within **each layer**
- Let  $x \in \mathbb{R}^d$  be an individual vector in the model
- Let  $\mu = \sum_{j=1}^d x_j$ , this is the mean
- Let  $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^d (x_j - \mu)^2}$ , this is the standard deviation
- Let  $\gamma, \beta \in \mathbb{R}^d$  be learned gain and bias parameters

Then LayerNorm does:

$$\text{output} = \frac{x - \mu}{\sigma + \epsilon}$$

# Layer normalization [Ba et al., 2016]

- **Layer normalization** is a trick to help models train faster
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation within **each layer**
- Let  $x \in \mathbb{R}^d$  be an individual vector in the model
- Let  $\mu = \sum_{j=1}^d x_j$ , this the mean
- Let  $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^d (x_j - \mu)^2}$ , this is the standard deviation
- Let  $\gamma, \beta \in \mathbb{R}^d$  be learned gain and bias parameters

Then LayerNorm does:

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

# Scaling the product attention [Vasvani et al., 2017]

- **Scaled Dot Product** attention is a final variation to aid in Transformer training
- When dimensionality  $d$  becomes large, dot products between vectors tend to become large.

# Scaling the product attention [Vasvani et al., 2017]

- **Scaled Dot Product** attention is a final variation to aid in Transformer training
- When dimensionality  $d$  becomes large, dot products between vectors tend to become large.
  - Because of this, inputs to the softmax function can be large, making the gradients small.

# Scaling the product attention [Vasvani et al., 2017]

- **Scaled Dot Product** attention is a final variation to aid in Transformer training
- When dimensionality  $d$  becomes large, dot products between vectors tend to become large.
  - Because of this, inputs to the softmax function can be large, making the gradients small.
- Instead of the self-attention function we have seen:

$$\text{output}_l = \text{softmax}(XQ_lK_l^T X^T)XV_l$$

# Scaling the product attention [Vasvani et al., 2017]

- **Scaled Dot Product** attention is a final variation to aid in Transformer training
- When dimensionality  $d$  becomes large, dot products between vectors tend to become large.
  - Because of this, inputs to the softmax function can be large, making the gradients small.
- Instead of the self-attention function we have seen:

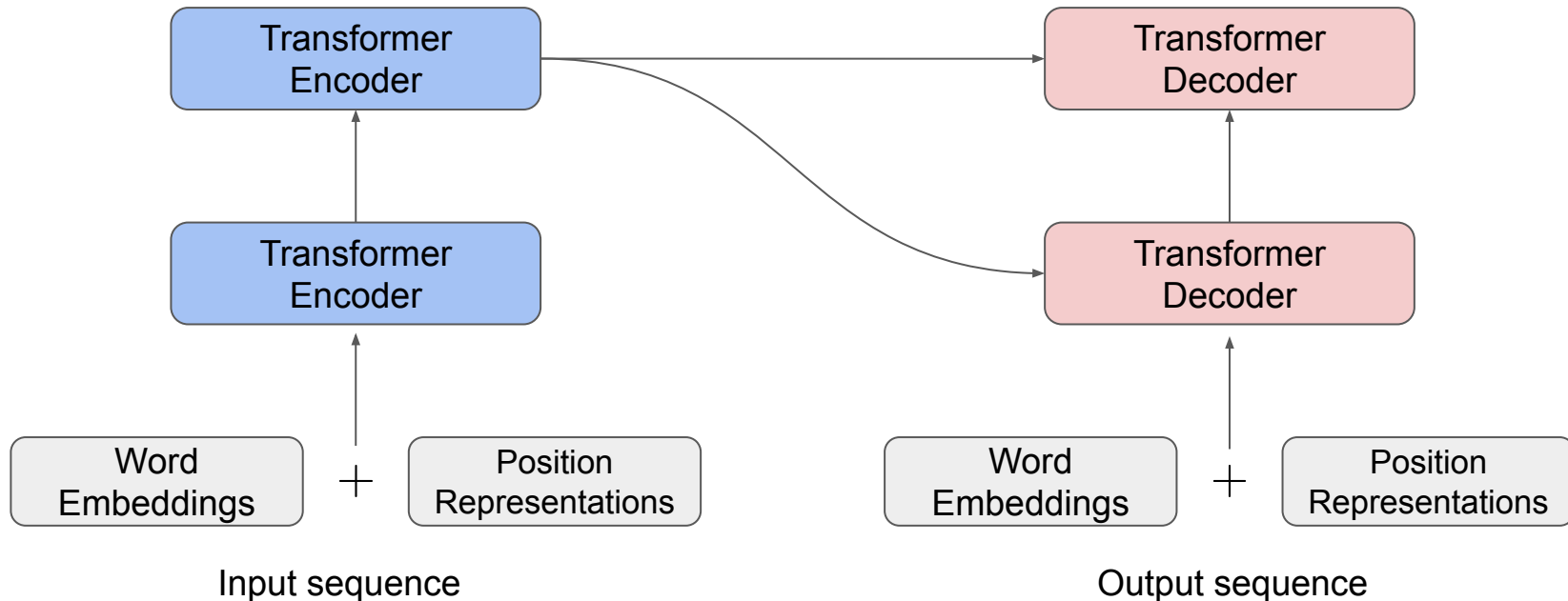
$$\text{output}_l = \text{softmax}(XQ_lK_l^T X^T)XV_l$$

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

$$\text{output}_l = \text{softmax}\left(\frac{XQ_lK_l^T X^T}{\sqrt{dh}}\right)XV_l$$

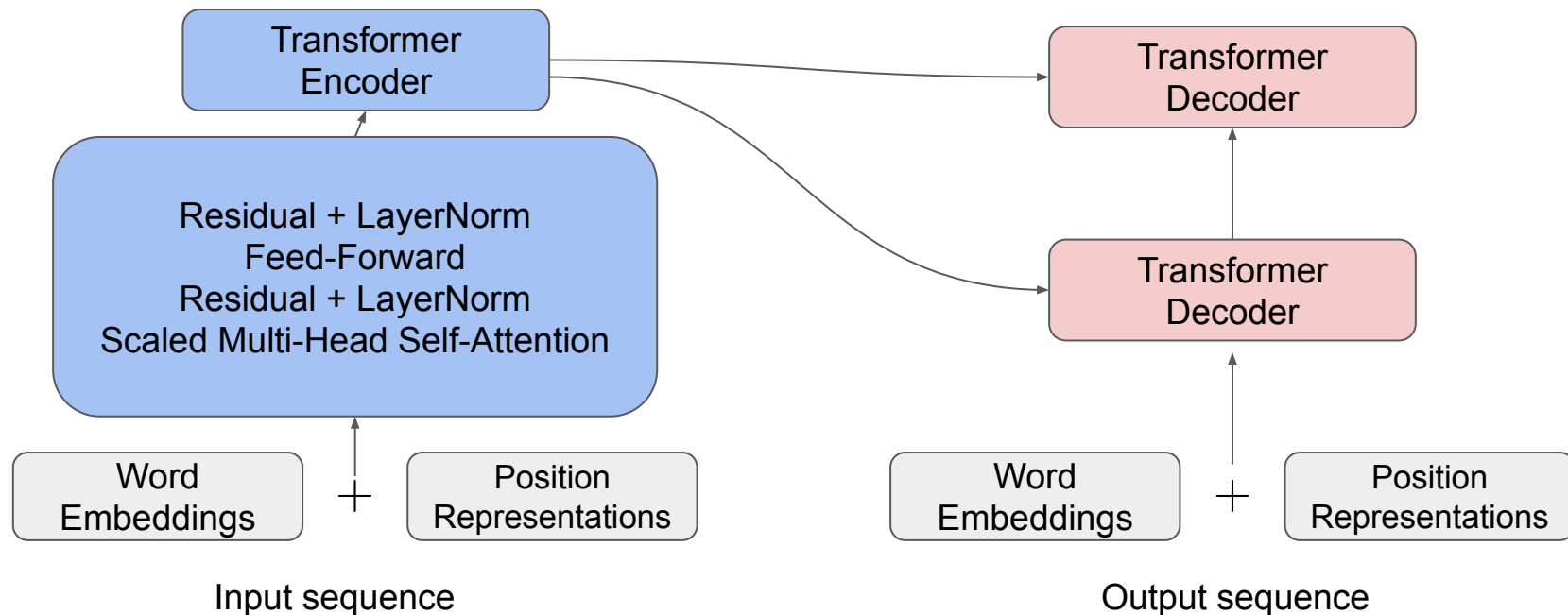
# Transformer Encoder-Decoder

Let's look at the Transformer Encoder and Decoder Blocks at a high level.



# Transformer Encoder-Decoder

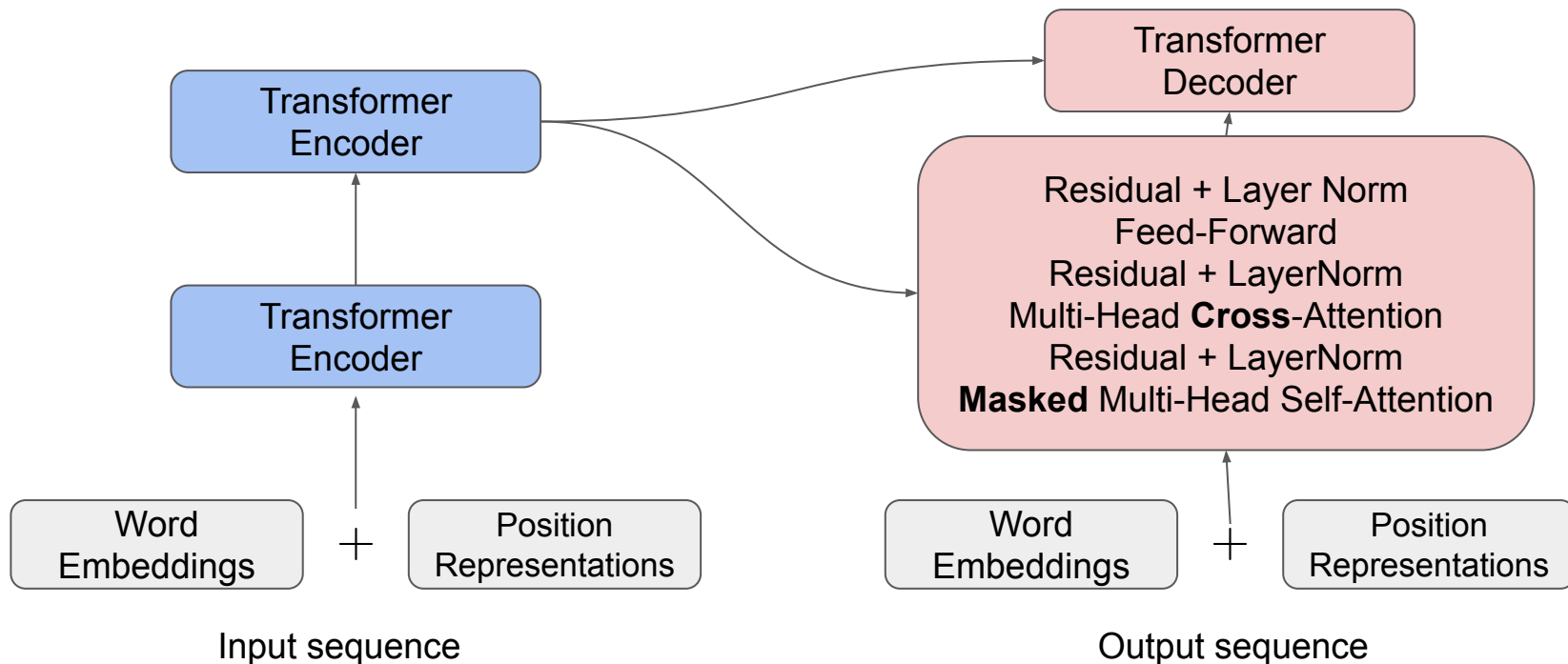
Let's look at the Transformer Encoder and Decoder Blocks at a high level.





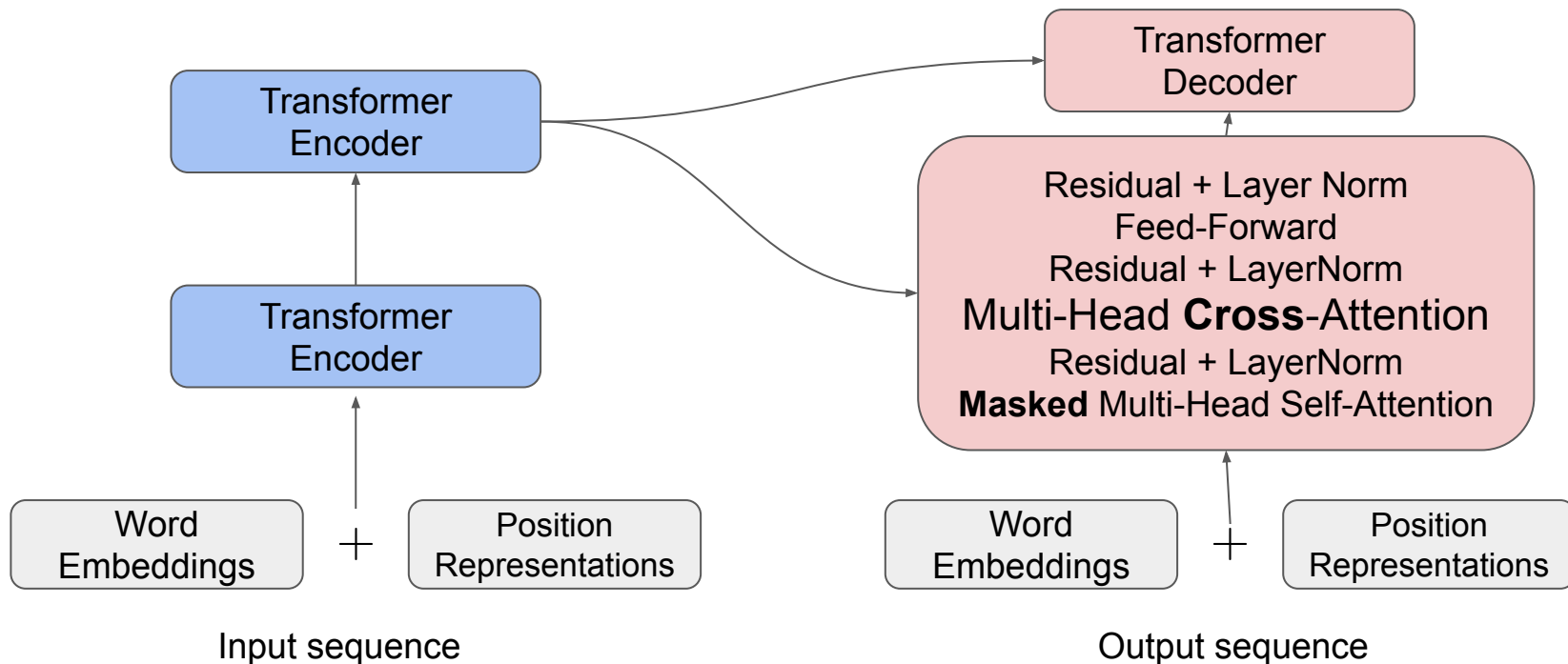
# Transformer Encoder-Decoder

Let's look at the Transformer Encoder and Decoder Blocks at a high level.



# Transformer Encoder-Decoder

Let's look at the Transformer Encoder and Decoder Blocks at a high level.



# Cross-attention

We saw that self-attention is when keys, queries, and values come from the same source.

Let  $h_1, \dots, h_T$  be **output** vectors from the last Transformer **encoder block**

Let  $z_1, \dots, z_T$  be **input** vectors from the Transformer **decoder**

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,  $q_i = Qz_i$

# Cross-attention

Let's look at how cross-attention is computed in matrices.

Let  $H = [h_1; \dots; h_T] \in \mathbb{R}^{T \times d}$  be the concatenation of encoder vectors

Let  $Z = [z_1; \dots; z_T] \in \mathbb{R}^{T \times d}$  be the concatenation of decoder vector

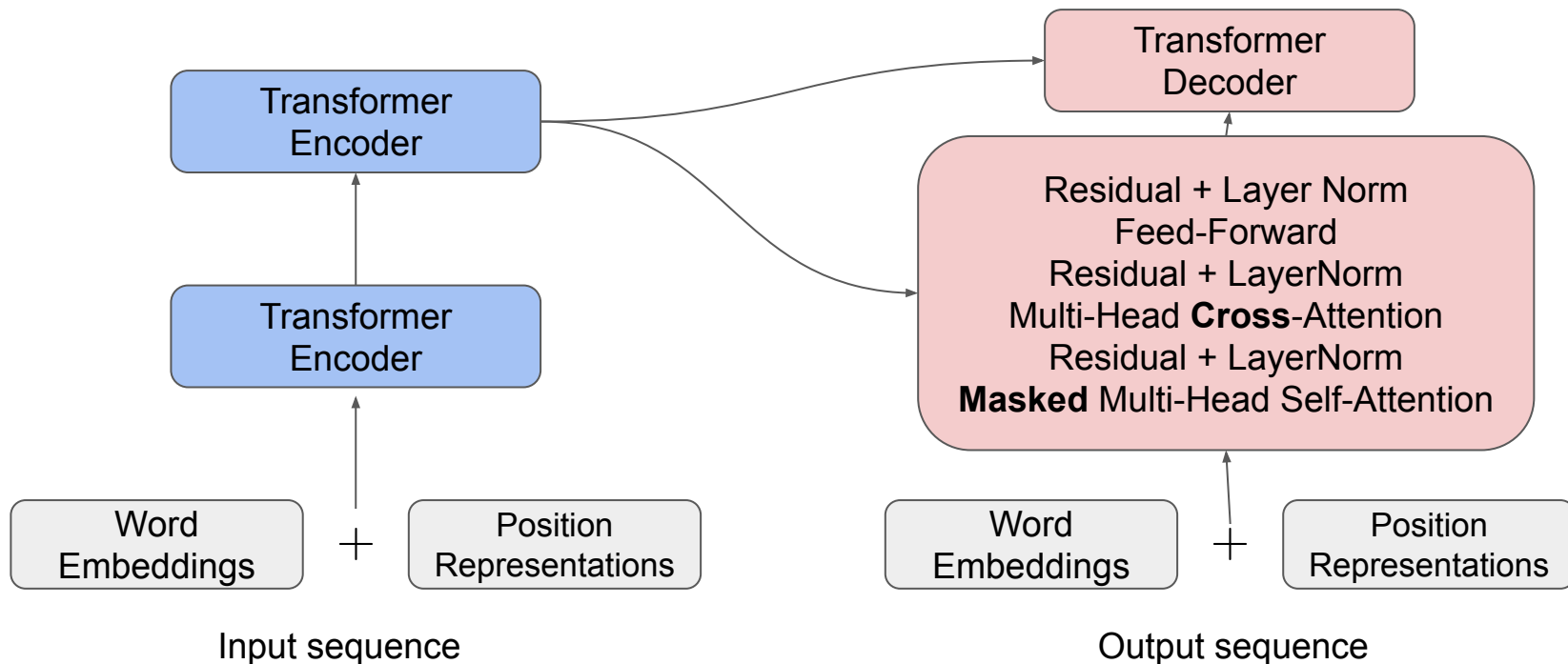
The output is defined as  $\text{output} = \text{softmax}(ZQ(HK)^T)HV$

First, take the query-key dot products in one matrix multiplication

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

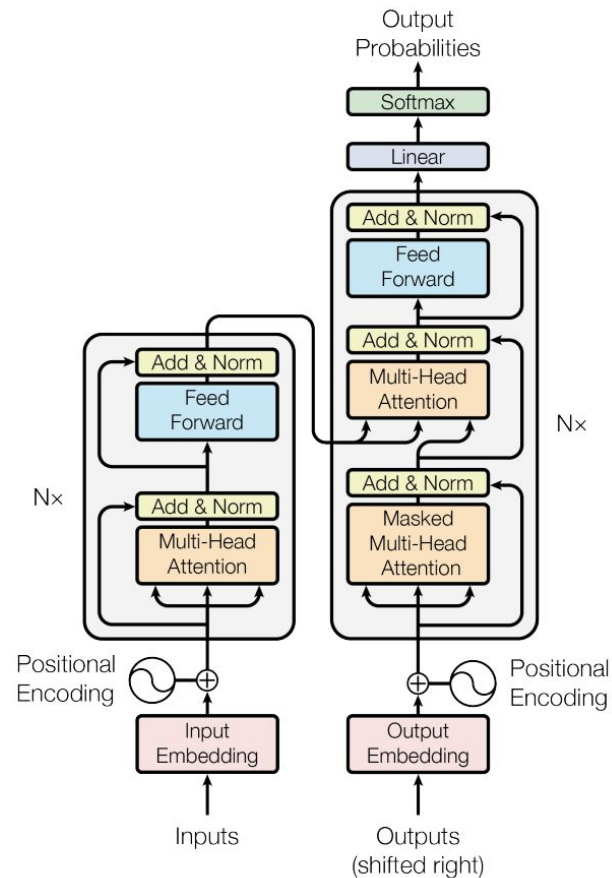
# Transformer Encoder-Decoder

Let's look at the Transformer Encoder and Decoder Blocks at a high level.



# Transformer Encoder-Decoder

[Vasvani et al., 2017]



# Great Results with Machine Translation

# Great Results with Machine Translation

Table 2: The Transformer achieves better BLEU scores than previous English-to-German and English-to-French newstest2014 tests at a 1

Model	BLEU	
	EN-DE	EN-FR
ByteNet [18]	23.75	
Deep-Att + PosUnk [39]		39.2
GNMT + RL [38]	24.6	39.92
ConvS2S [9]	25.16	40.46
MoE [32]	26.03	40.56
Deep-Att + PosUnk Ensemble [39]		40.4
GNMT + RL Ensemble [38]	26.30	41.16
ConvS2S Ensemble [9]	26.36	<b>41.29</b>
Transformer (base model)	27.3	38.1
Transformer (big)	<b>28.4</b>	<b>41.8</b>



## But also efficient training

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

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	<b>41.29</b>	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	<b><math>3.3 \cdot 10^{18}</math></b>	
Transformer (big)	<b>28.4</b>	<b>41.8</b>	$2.3 \cdot 10^{19}$	

# Document generation [Liu et al., 2018]

Table 4: Performance of best models of each model architecture using the combined corpus and tf-idf extractor.

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

# Constituency parsing

Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser et al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser et al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

# Attention perks

## Attention Visualizations

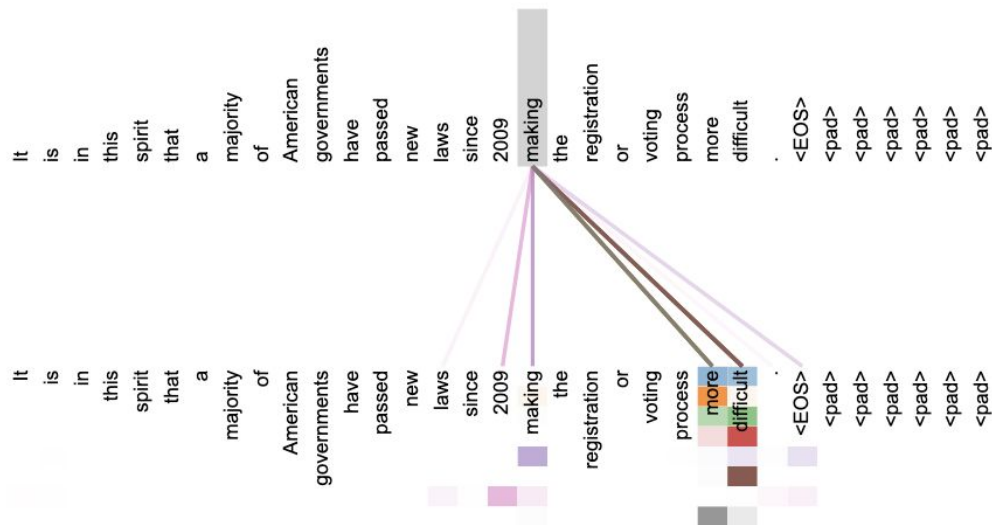
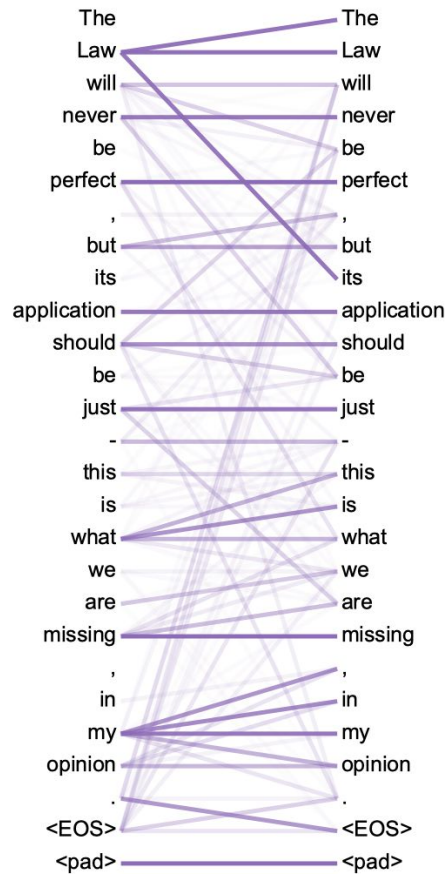


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

# Attention perks













# Transformers are taking the field

Before too long, most Transformers results also included **pretraining**, a method we'll go over at next lecture.

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

# GLUE benchmark

Rank Name		Model	URL	Score	CoLA	SST-2	MRPC	STS-B	QQP
1	T5 Team - Google	T5		89.7	70.8	97.1	91.9/89.2	92.5/92.1	74.6/90.4
2	ALBERT-Team Google Language	ALBERT (Ensemble)		89.4	69.1	97.1	93.4/91.2	92.5/92.0	74.2/90.5
+	3 王玮	ALICE v2 large ensemble (Alibaba DAMO NLP)		89.0	69.2	97.1	93.6/91.5	92.7/92.3	74.4/90.7
4	Microsoft D365 AI & UMD	FreeLB-RoBERTa (ensemble)		88.8	68.0	96.8	93.1/90.8	92.4/92.2	74.8/90.3
5	Facebook AI	RoBERTa		88.5	67.8	96.7	92.3/89.8	92.2/91.9	74.3/90.2
6	XLNet Team	XLNet-Large (ensemble)		88.4	67.8	96.8	93.0/90.7	91.6/91.1	74.2/90.3
+	7 Microsoft D365 AI & MSR AI	MT-DNN-ensemble		87.6	68.4	96.5	92.7/90.3	91.1/90.7	73.7/89.9
8	GLUE Human Baselines	GLUE Human Baselines		87.1	66.4	97.8	86.3/80.8	92.7/92.6	59.5/80.4
9	Stanford Hazy Research	Snorkel MeTaL		83.2	63.8	96.2	91.5/88.5	90.1/89.7	73.1/89.9
10	XLM Systems	XLM (English only)		83.1	62.9	95.6	90.7/87.1	88.8/88.2	73.2/89.8

Transformers - don't stop the journey



# Transformers - don't stop the journey

- **Position representations:**
  - Are simple absolute indices the best we can do to represent position?
  - Relative linear position attention [Shaw et al., 2018]

# Transformers - don't stop the journey

- **Position representations:**
  - Are simple absolute indices the best we can do to represent position?
  - Relative linear position attention [Shaw et al., 2018]
- **Quadratic compute in self-attention:**
  - Computing all pairs of interactions means our computation grows quadratically with the sequence length
  - For recurrent models, it only grew linearly

# Quadratic computation as a function of sequence length

- One of the benefits of self-attention over recurrence was that it's highly parallelizable
- However, its total number of operations grows as  $O(T^2 d)$ , where  $T$  is the sequence lengths, and  $d$  is the dimensionality.

Think of  $d$  as around 1,000.

- So for a single sentence  $T < 30$ ,  $T^2 < 900$
- In practice we set a bound like  $T = 512$
- What if we would like to  $T > 1000$  such as long documents?

$$\boxed{XQ} \boxed{K^T X^T} = \boxed{XQK^T X^T} \in \mathbb{R}^{T \times T}$$

# Improving on quadratic self-attention cost?

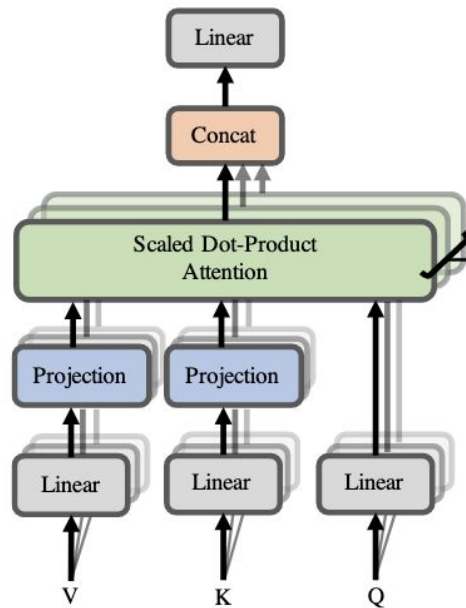
Considerable recent work has gone into the question can we build models without paying the  $O(T^2)$  all pairs self-attention cost?

# Improving on quadratic self-attention cost?

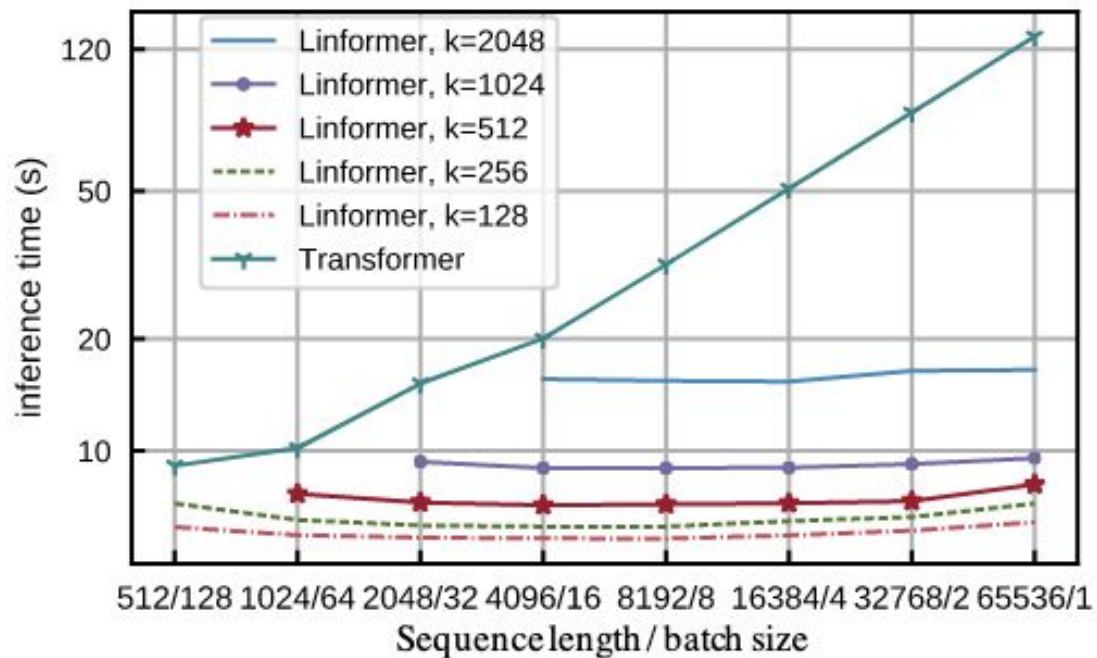
Considerable recent work has gone into the question can we build models without paying the  $O(T^2)$  all pairs self-attention cost?

Linformer [Wang et al 2020] proposes one idea.

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



# Improving on quadratic self-attention cost?



# 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**?

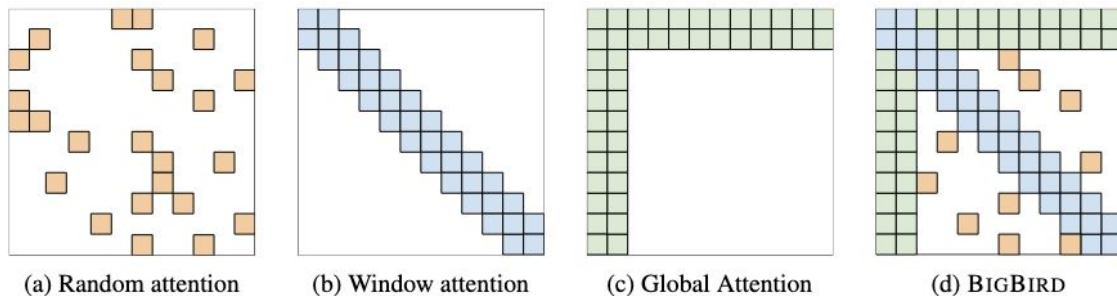


Figure 1: Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with  $r = 2$ , (b) sliding window attention with  $w = 3$  (c) global attention with  $g = 2$ . (d) the combined BIGBIRD model.

# 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**?

Model	HotpotQA			NaturalQ		TriviaQA	WikiHop
	Ans	Sup	Joint	LA	SA	Full	MCQ
RoBERTa	73.5	83.4	63.5	-	-	74.3	72.4
Longformer	74.3	84.4	64.4	-	-	75.2	75.0
BIGBIRD-ITC	<b>75.7</b>	86.8	67.7	70.8	53.3	<b>79.5</b>	<b>75.9</b>
BIGBIRD-ETC	75.5	<b>87.1</b>	<b>67.8</b>	<b>73.9</b>	<b>54.9</b>	78.7	<b>75.9</b>

Table 2: QA Dev results using Base size models. We report accuracy for WikiHop and F1 for HotpotQA, Natural Questions, and TriviaQA.



# Performer [Choromański et al. 2021]

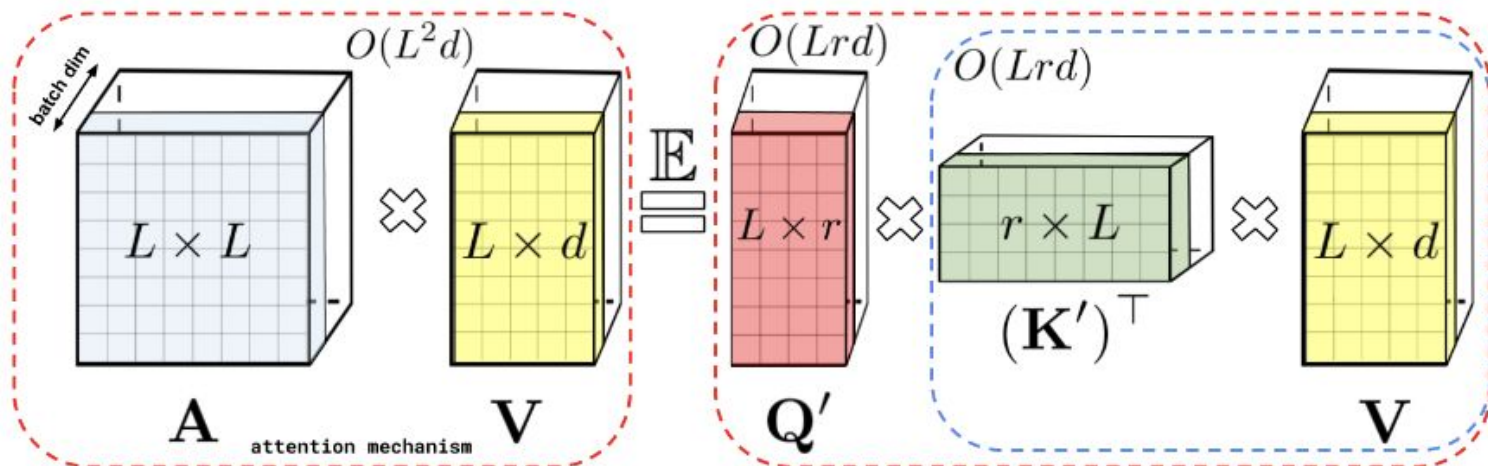


Figure 1: Approximation of the regular attention mechanism  $\mathbf{A}\mathbf{V}$  (before  $\mathbf{D}^{-1}$ -renormalization) via (random) feature maps. Dashed-blocks indicate order of computation with corresponding time complexities attached.

# Performer [Choromański et al. 2021]

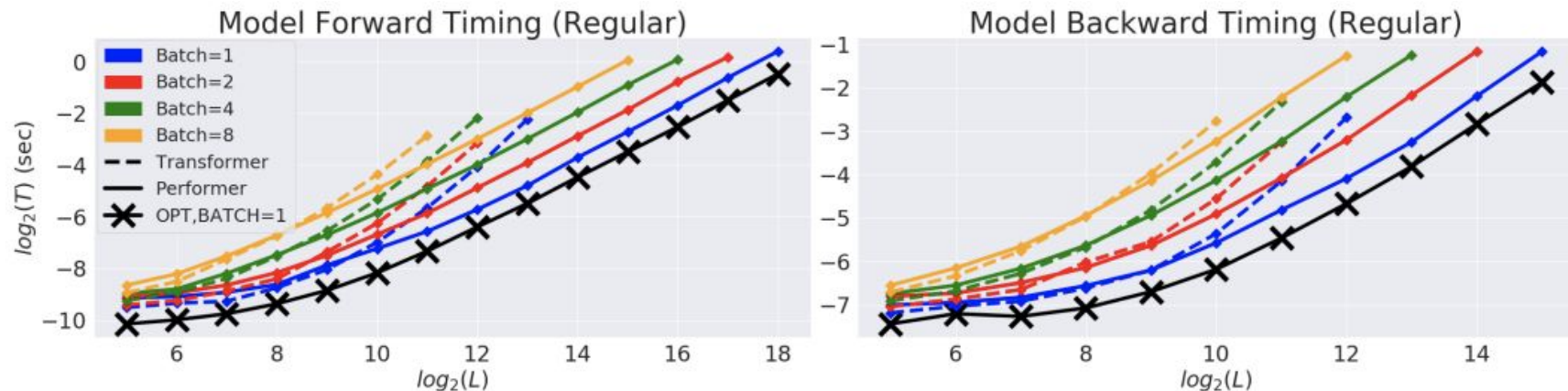


Figure 3: Comparison of Transformer and Performer in terms of forward and backward pass speed and maximum  $L$  allowed. "X" (OPT) denotes the maximum possible speedup achievable, when attention simply returns the  $V$ -matrix. Plots shown up to when a model produces an out of memory error on a V100 GPU with 16GB. Vocabulary size used was 256. Best in color.

# Sparse Transformers [Jaszczur et al., 2022]

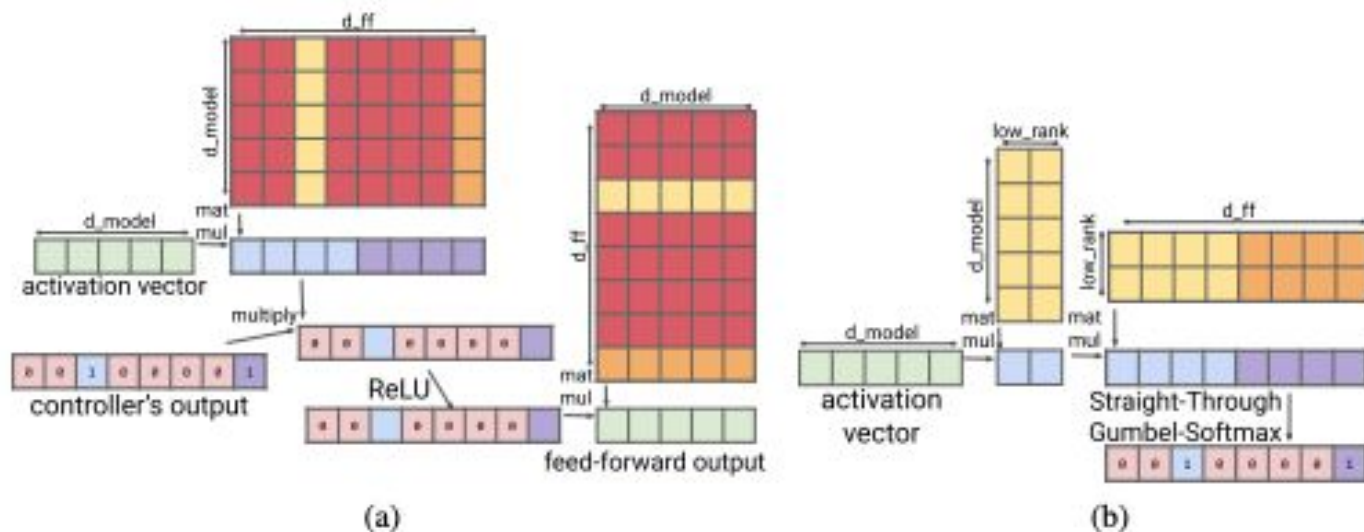


Figure 2: (a) Sparse Feedforward Layer only activates 1 in  $N$  rows/columns of each block to reduce the decoding time. Here only two rows/columns in blocks of size 4 are loaded while the weights in dark red are not loaded from memory during inference. (b) Sparse Feedforward Controller with the output of 2 blocks of size 4 (1 in 4 sparsity).

# Sparse Transformers [Jaszczur et al., 2022]

	Params	Dec. time	Dec. time per block
baseline Transf.	800M	0.160s	5.9ms
+ Sparse FF	-	0.093s	3.1ms
+ Sparse QKV	-	0.152s	6.2ms
+ Sparse FF+QKV	-	0.061s	1.9ms
Speedup		2.62x	3.05x
baseline Transf.	17B	3.690s	0.581s
+Sparse FF	-	1.595s	0.259s
+Sparse QKV	-	3.154s	0.554s
+Sparse FF+QKV	-	0.183s	0.014s
Speedup		20.0x	42.5x

# Literature

1. He et al., 2015 - <https://arxiv.org/pdf/1512.03385v1.pdf>
2. Ba et al., 2016 - <https://arxiv.org/abs/1607.06450>
3. Vasvani et al., 2017 - <https://arxiv.org/pdf/1706.03762.pdf>
4. Wang et al., 2020 - <https://arxiv.org/pdf/2006.04768.pdf>
5. Zaheer et al., 2020 - <https://arxiv.org/pdf/2007.14062.pdf>
6. Choromański et al., 2020 - <https://arxiv.org/abs/2009.14794>
7. Jaszczur et al., 2021 - <https://proceedings.neurips.cc/paper/2021/file/51f15efdd170e6043fa02a74882f0470-Paper.pdf>

Thanks!