

CSE 156 | Lecture 7: Self-Attention & Transformers

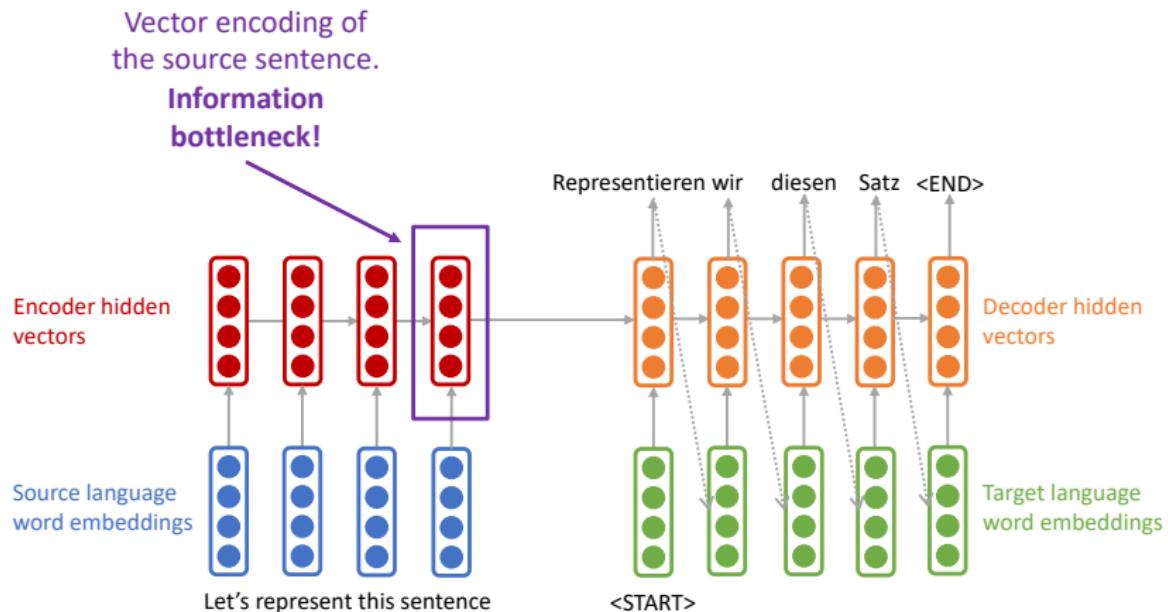
Ndapa Nakashole

October 22, 2024

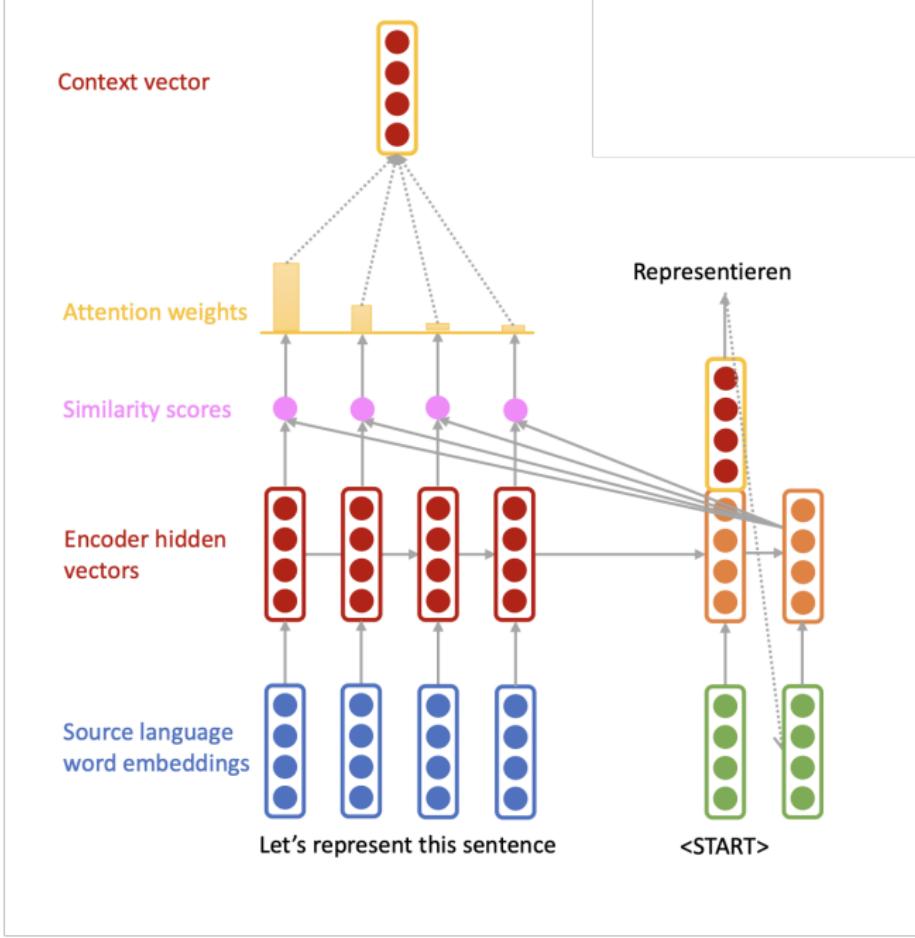
Administrative matters

- ▶ **PA2:** out, due Nov 06 (no deadline extensions - start early!)
- ▶ **Quiz 1:** out Friday, next Wednesday Oct 30th
- ▶ **Communicating with us:** As mentioned in Lecture 1, Piazza is the best way to reach us. Please refrain from sending emails to the instructor or TAs. We want to ensure **uniform and timely responses** to all students.
 - **Public posts** for class-related questions
 - **Private posts** for person-specific matters, e.g., grades, etc.
 - For **truly private** matters, only then email the instructor.

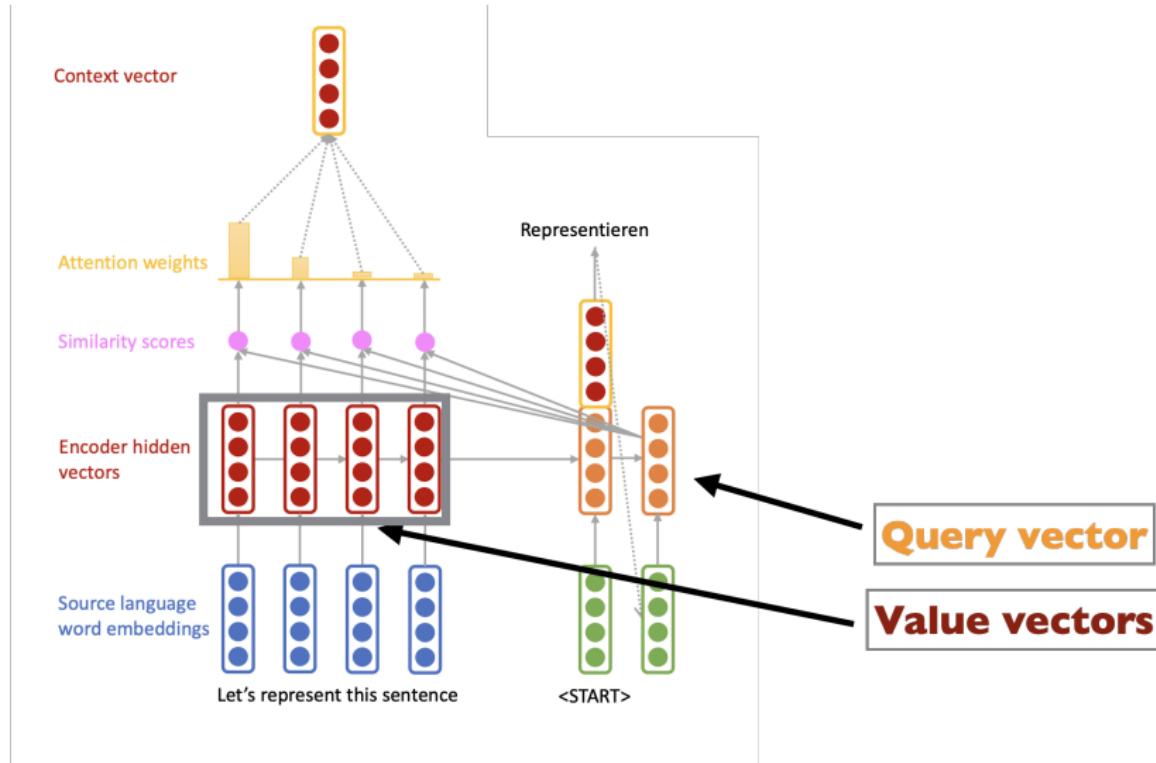
Recap: Seq2Seq



Recap: Seq2Seq with Attention

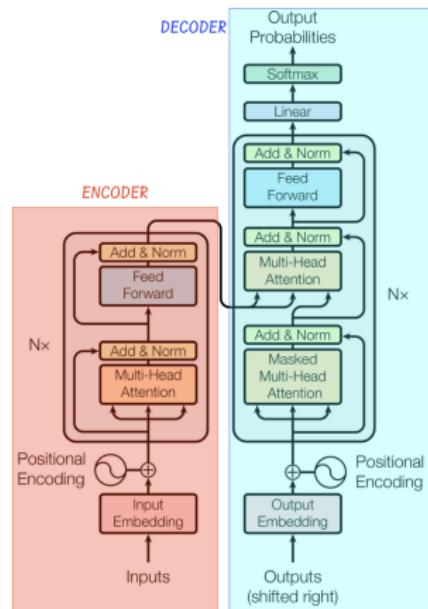


Recap: Seq2Seq with Attention, Terminology



Attention is All you Need

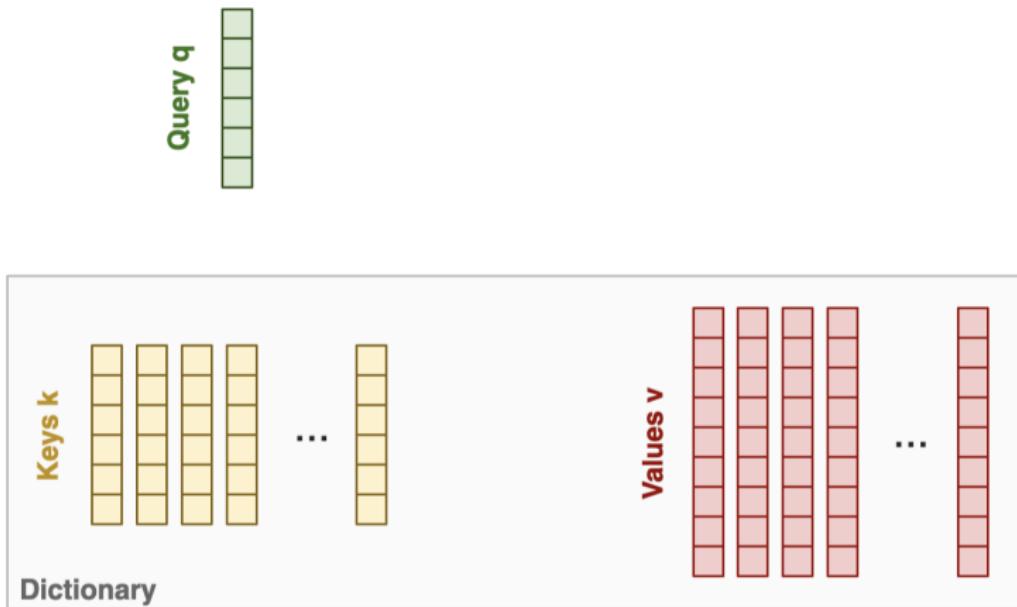
- ▶ Attention
 - Self-attention
 - Interspersing of Attention and Feedforward layers
 - Multiple attention heads in parallel
 - Encoder-Decoder Attention (cross-attention)
- ▶ Positional Encoding (attention is order-agnostic)
- ▶ Optimization Tricks
 - Residual network (ResNet) structure
 - LayerNorms
 - Scaled dot-product attention



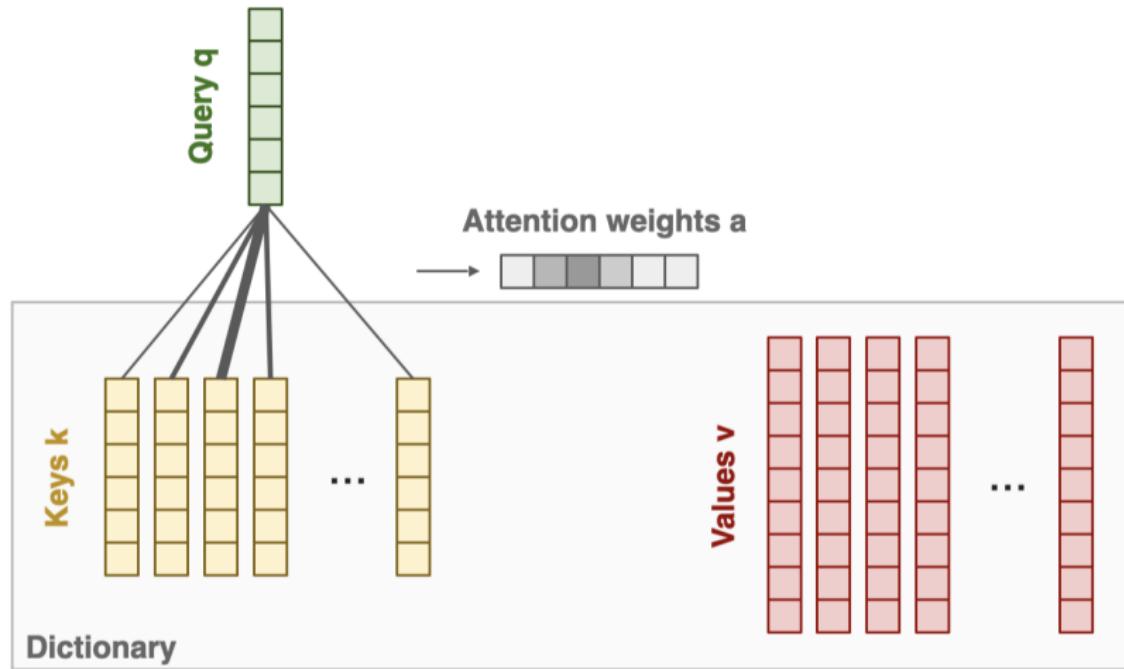
Self-Attention: Overview

- ▶ **Given:** Query vector q and a set of Key-Value ($k-v$) vector pairs.
- ▶ **Output:** a weighted sum of the **values**, where the weights are determined by the similarity of the **query** to the **keys**.
 - A selective summary of the **values**, with respect to the **query**.
- ▶ **Query, Keys, Values**, and **Outputs** are all vectors.

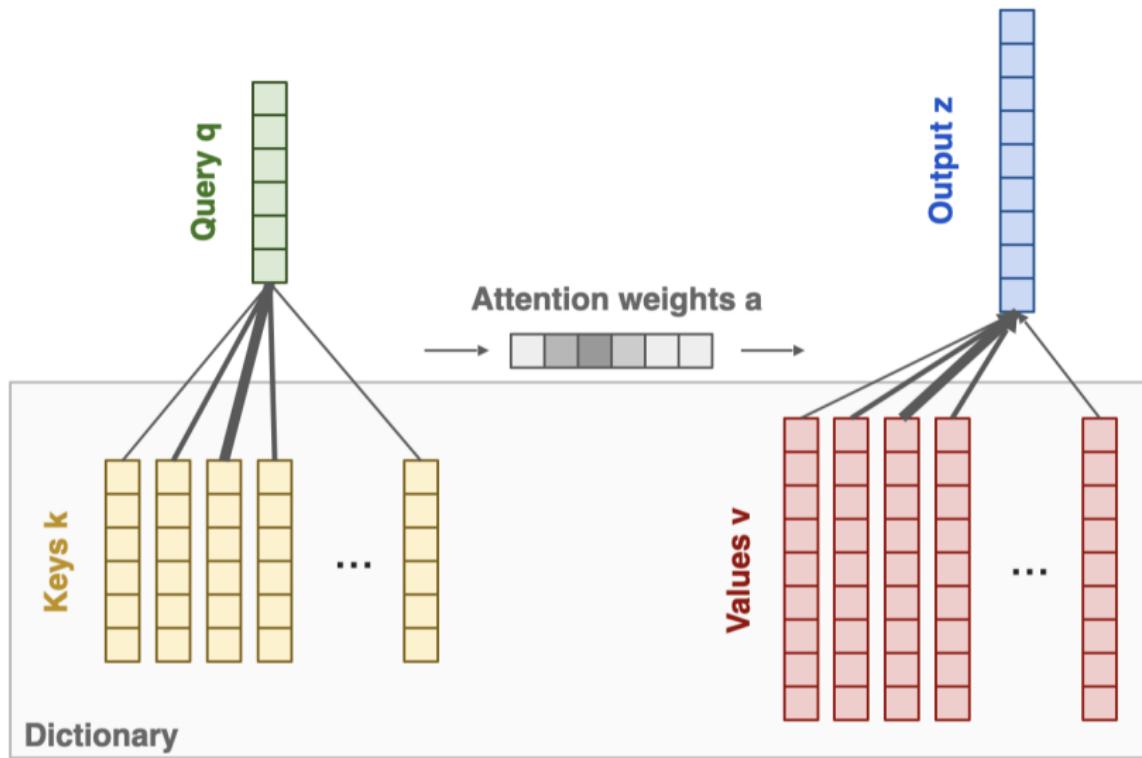
Queries, Keys, & Values: Example



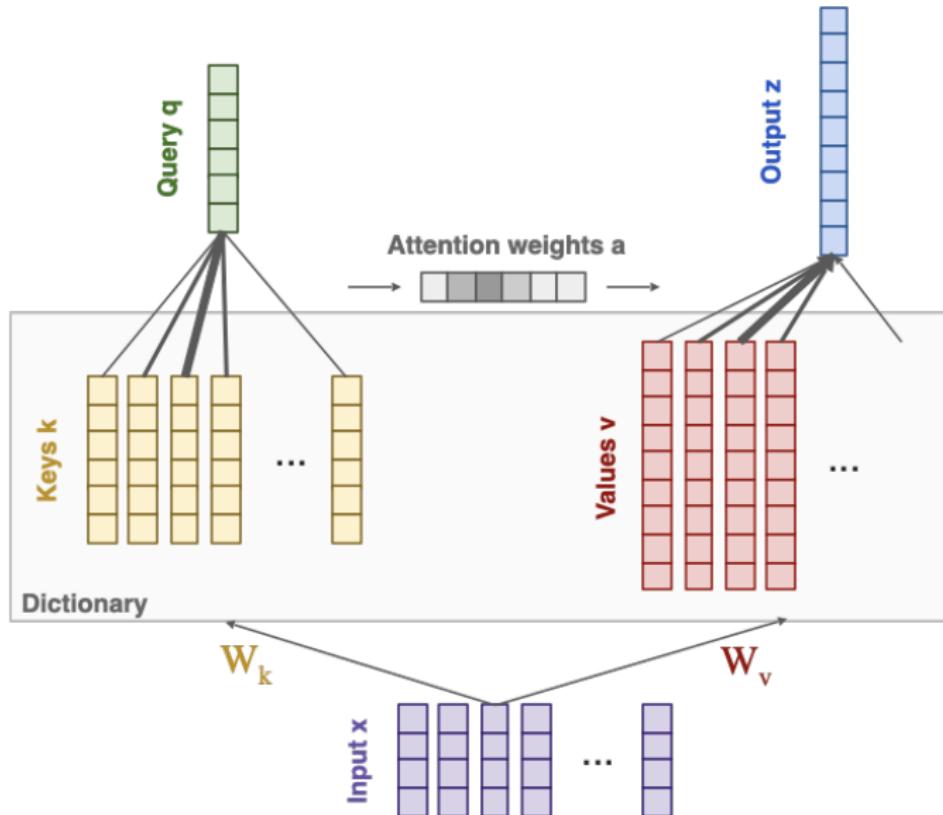
Computing the Attention Scores



Output



Where do the Keys, Queries, and Values come from?



Key, Value, and Query Derivation

\mathbf{k} and \mathbf{v} are derived from the same input \mathbf{x} :

$$\mathbf{k} = \mathbf{W}_k \cdot \mathbf{x} \quad \mathbf{v} = \mathbf{W}_v \cdot \mathbf{x}$$

The query \mathbf{q} can come from a separate input \mathbf{y} :

$$\mathbf{q} = \mathbf{W}_q \cdot \mathbf{y}$$

Or from the same input \mathbf{x} , then we call it "self-attention":

$$\mathbf{q} = \mathbf{W}_q \cdot \mathbf{x}$$

Attention Mechanism: 1 Query

$$\text{similarity}_i = \mathbf{q} \cdot \mathbf{k}_i$$

$$\text{attention}_i = \frac{e^{\mathbf{q} \cdot \mathbf{k}_i}}{\sum_j e^{\mathbf{q} \cdot \mathbf{k}_j}}$$

Compute the Weighted Sum of the **Values**:

$$A(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \sum_i \text{attention}_i \cdot \mathbf{v}_i$$

Expands to:

$$A(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \sum_i \frac{e^{\mathbf{q} \cdot \mathbf{k}_i}}{\sum_j e^{\mathbf{q} \cdot \mathbf{k}_j}} \mathbf{v}_i$$

where $A \in \mathbb{R}^{d_v}$

Example Setup

Given:

- ▶ **Query** vector $\mathbf{q} \in \mathbb{R}^{1 \times d_k}$ (where $|Q| = 1$):

$$\mathbf{q} = [2 \quad 1]$$

- ▶ **Key** matrix $\mathbf{K} \in \mathbb{R}^{|K| \times d_k}$ (where $d_k = 2$ and $|K| = 2$):

$$\mathbf{K} = [\mathbf{k}_1 \quad \mathbf{k}_2] = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

- ▶ **Value** matrix $\mathbf{V} \in \mathbb{R}^{|K| \times d_v}$ (where $|K| = 2$ and $d_v = 2$):

$$\mathbf{V} = \begin{bmatrix} 3 & 7 \\ 6 & 12 \end{bmatrix}$$

Step 1: Compute Similarity Scores then apply Softmax

We compute the dot product between the **query** vector \mathbf{q} and each **key** vector \mathbf{k}_i :

$$\text{similarity}_1 = \mathbf{q} \cdot \mathbf{k}_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 2$$

$$\text{similarity}_2 = \mathbf{q} \cdot \mathbf{k}_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 3$$

Next, we apply the softmax function to the similarity scores:

$$\text{attention}_1 = \frac{e^2}{e^2 + e^3} = \frac{7.389}{7.389 + 20.086} = \frac{7.389}{27.475} \approx 0.269$$

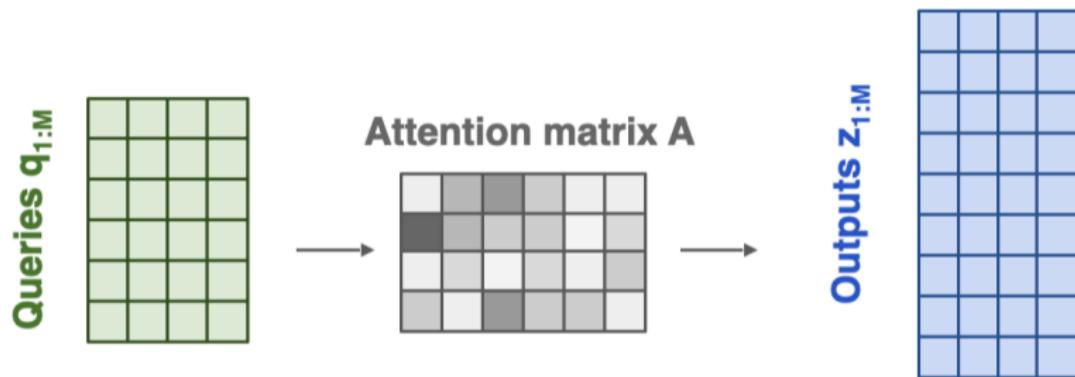
$$\text{attention}_2 = \frac{e^3}{e^2 + e^3} = \frac{20.086}{7.389 + 20.086} = \frac{20.086}{27.475} \approx 0.731$$

Step 2: Compute Weighted Sum of Values

$$\begin{aligned} A(\mathbf{q}, \mathbf{K}, \mathbf{V}) &= \text{attention}_1 \cdot \mathbf{v}_1 + \text{attention}_2 \cdot \mathbf{v}_2 \\ &= 0.269 \cdot \begin{bmatrix} 3 \\ 6 \end{bmatrix} + 0.731 \cdot \begin{bmatrix} 7 \\ 12 \end{bmatrix} \\ &= \begin{bmatrix} 0.807 \\ 1.614 \end{bmatrix} + \begin{bmatrix} 5.117 \\ 8.772 \end{bmatrix} \\ &= \begin{bmatrix} 5.924 \\ 10.386 \end{bmatrix} \end{aligned}$$

Matrix Notation: computing attention scores for multiple queries at once

Stacked queries --> Attention matrix --> Stacked outputs:



$$\text{Attention matrix} = \text{softmax}(\mathbf{Q} \cdot \mathbf{K}^T)$$

$$\text{Output} = \text{softmax}(\mathbf{Q} \cdot \mathbf{K}^T) \cdot \mathbf{V}$$

Attention Mechanism: Matrix Formulation

Stack **queries** in a matrix \mathbf{Q} : $[|Q| \times d_k]$ to compute attention matrix \mathbf{A} , and then multiply by **values** \mathbf{V} :

$$A(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}^\top) \cdot \mathbf{V}$$

$[|Q| \times d_k] \times [d_k \times |K|] \times [|K| \times d_v]$

Output: $|Q| \times d_v$

- ▶ Problem: scale of dot product increases as dimensions get larger
- ▶ Fix: scale by size of the vector

Scaled dot-product :

$$\left(\mathbf{Q}\mathbf{K}^\top / \sqrt{d_k} \right)$$

Example Setup

Given:

- ▶ **Query** matrix $\mathbf{Q} \in \mathbb{R}^{|Q| \times d_k}$ (where $|Q| = 3, d_k = 2$):

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 1 \end{bmatrix}$$

- ▶ **Key** matrix $\mathbf{K} \in \mathbb{R}^{|K| \times d_k}$ (where $|K| = 2, d_k = 2$):

$$\mathbf{K} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

- ▶ **Value** matrix $\mathbf{V} \in \mathbb{R}^{|K| \times d_v}$ (where $|K| = 2, d_v = 2$):

$$\mathbf{V} = \begin{bmatrix} 4 & 8 \\ 6 & 12 \end{bmatrix}$$

Step 1: Compute Attention Weights

$$\mathbf{Q} \cdot \mathbf{K}^\top = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^T = \begin{bmatrix} 1 & 0 \\ 3 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\text{attention}_{1,1} = \frac{e^1}{e^1 + e^0} \approx 0.731, \quad \text{attention}_{1,2} = \frac{e^0}{e^1 + e^0} \approx 0.269$$

$$\text{attention}_{2,1} = \frac{e^3}{e^3 + e^1} \approx 0.880, \quad \text{attention}_{2,2} = \frac{e^1}{e^3 + e^1} \approx 0.120$$

$$\text{attention}_{3,1} = \frac{e^1}{e^1 + e^1} = 0.5, \quad \text{attention}_{3,2} = \frac{e^1}{e^1 + e^1} = 0.5$$

$$\text{softmax}(\mathbf{Q} \cdot \mathbf{K}^\top) = \begin{bmatrix} 0.731 & 0.269 \\ 0.880 & 0.120 \\ 0.5 & 0.5 \end{bmatrix}$$

Step 2: Compute Weighted Sum of Values

$$A(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q} \cdot \mathbf{K}^\top) \cdot \mathbf{V}$$

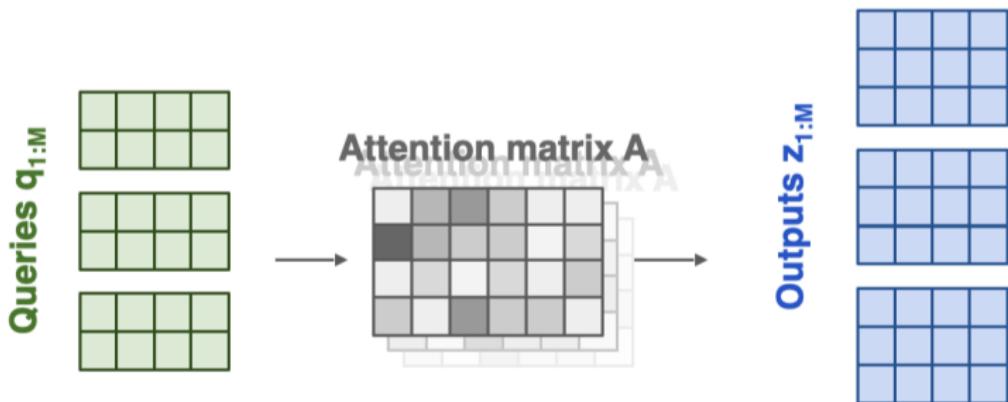
$$A(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \begin{bmatrix} 0.731 & 0.269 \\ 0.880 & 0.120 \\ 0.5 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} 4 & 8 \\ 6 & 12 \end{bmatrix}$$

$$\begin{aligned} A(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \begin{bmatrix} 0.731 \cdot 4 + 0.269 \cdot 6 & 0.731 \cdot 8 + 0.269 \cdot 12 \\ 0.880 \cdot 4 + 0.120 \cdot 6 & 0.880 \cdot 8 + 0.120 \cdot 12 \\ 0.5 \cdot 4 + 0.5 \cdot 6 & 0.5 \cdot 8 + 0.5 \cdot 12 \end{bmatrix} \\ &= \begin{bmatrix} 4.538 & 9.076 \\ 4.240 & 8.480 \\ 5 & 10 \end{bmatrix} \end{aligned}$$

Thus, the output $A(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \in \mathbb{R}^{|Q| \times d_v}$, where $|Q| = 3$ and $d_v = 2$.

Multiple Heads

Repeat the process with different linear projections of Q, K, V :



Results are concatenated along the feature dimension.

Multi-Head Attention: Matrix Formulation

For each head:

$$A(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right) \cdot \mathbf{V}$$

- ▶ Each head computes: $\left[|Q| \times \frac{d_k}{h}\right] \times \left[\frac{d_k}{h} \times |K|\right] \times \left[|K| \times \frac{d_v}{h}\right]$
- ▶ Output per head: $|Q| \times \frac{d_v}{h}$

Concatenate the heads: h heads, concatenate their outputs:

$$\mathbf{O} = \text{Concat}(\mathbf{O}_1, \mathbf{O}_2, \dots, \mathbf{O}_h)$$

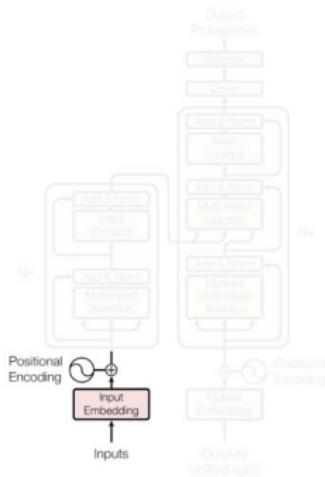
Where each $\mathbf{O}_i \in \mathbb{R}^{|Q| \times \frac{d_v}{h}}$.

Final output:

$$\mathbf{O} \in \mathbb{R}^{|Q| \times d_v}$$

Text Encoding in the Transformer

Input Processing



Input (Tokenization and) Embedding

Input text is first split into pieces. Can be characters, word, "tokens":

"The detective investigated" \rightarrow [The_] [detective_] [invest] [igat] [ed_]

Tokens are indices into the "vocabulary":

[The_] [detective_] [invest] [igat] [ed_] \rightarrow [3 721 68 1337 42]

Each vocab entry corresponds to a learned d_{model} -dimensional vector.

[3 721 68 1337 42] \rightarrow [[0.123, -5.234, ...], [...], [...], [...], [...]]

Positional Encoding

Remember attention is permutation invariant, but language is not!

("The mouse ate the cat" vs "The cat ate the mouse")

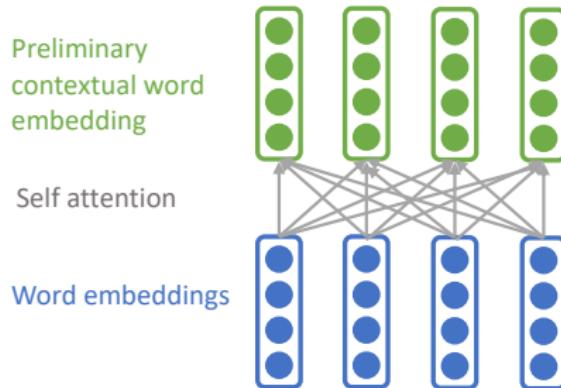
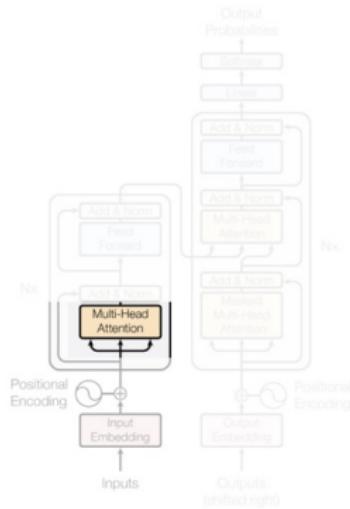
Need to encode position of each word; just add something.

Think [The_] + 10 [detective_] + 20 [invest] + 30 ... but smarter.

Attention: Communication Stage

Multi-headed Self-Attention

- ▶ Input sequence is used to create queries, keys, and values.
- ▶ Each token “looks around” the whole input, and decides how to update its representation based on what it sees.



Multi-Head Attention: Intuition

- ▶ Intuition: Information from different parts of the sentence can be useful to disambiguate in different ways.

I **run** a small business

I **run** a mile in 10 minutes

The robber made a **run** for it

The stocking had a **run**

- ▶ Each head can learn a different way to disambiguate (e.g., based on semantics, syntax, etc.)

Multi-Head Attention: Intuition

- ▶ Intuition: Information from different parts of the sentence can be useful to disambiguate in different ways.
- ▶ Syntax (nearby context)
- ▶ Semantics (farther context)

I run a small business

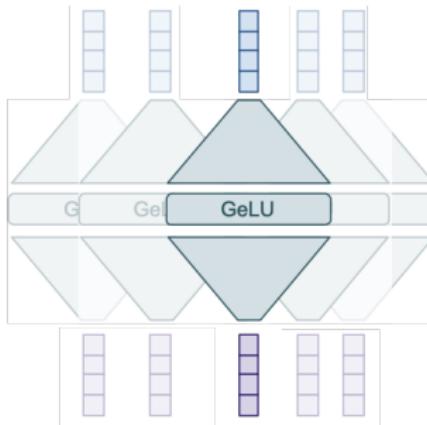
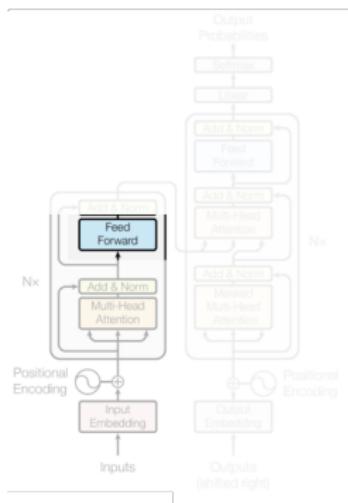
I run a mile in 10 minutes

The robber made a run for it

The stocking had a run

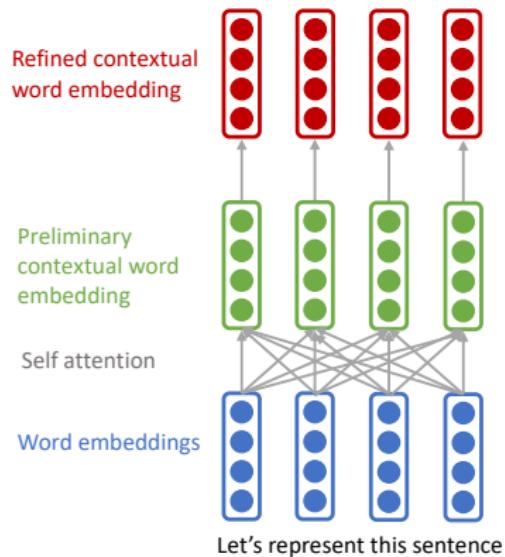
Feedforward Layers: Position-wise

- ▶ A simple MLP applied to each token individually: $z_i = W_2 \text{GeLU}(W_1 x_i + b_1) + b_2$
- ▶ Bulk of the parameters in giant models, this is what becomes giant.

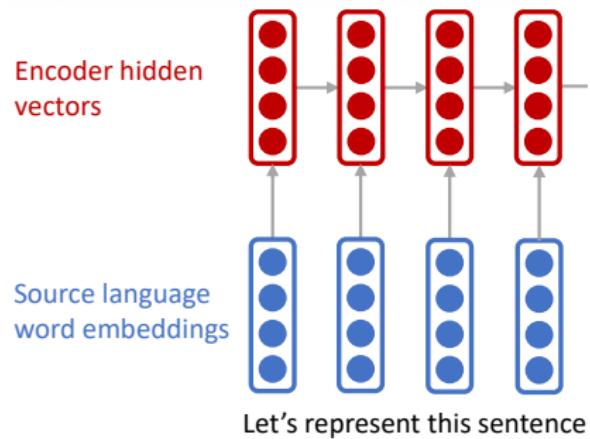


Encoder in a Transformer vs RNN

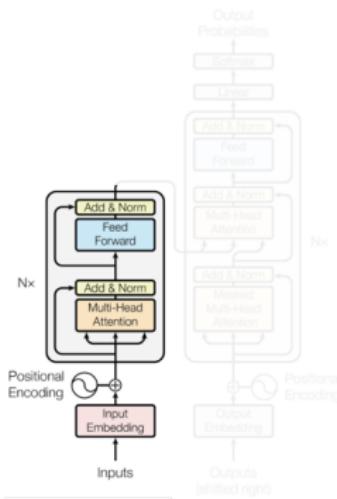
Transformer Encoder



RNN Encoder



The Encoder: Stacking Blocks



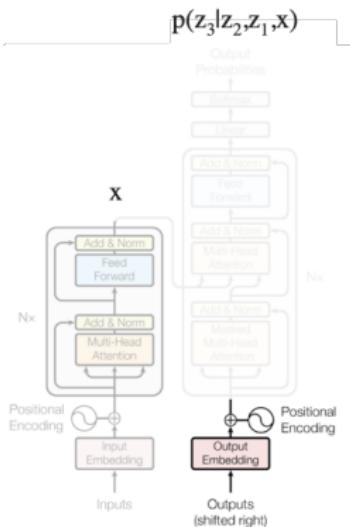
- ▶ Input & output shapes are identical, stack N such blocks.
- ▶ $N=6$ ("base"), $N=12$ ("large"), > 100 ("LLMs").
- ▶ Encoder output is a heavily processed, contextualized version of the input tokens, i.e. a sequence.
- ▶ The encoder output is used by the decoder to generate the output sequence.

Decoding with the Transformer

Decoding

- Want to model: $p(z | x)$ decompose as:

$$p(z | x) = p(z_1 | x) p(z_2 | z_1, x) p(z_3 | z_2, z_1, x) \dots$$



- Each p is a full pass through the model.
- For generating $p(z_3 | z_2, z_1, x)$:
 - x comes from the encoder,
 - z_1, z_2 is what we have predicted so far, goes into the decoder.
- Given: $p(z | x)$, still need to sample a sentence. Many strategies: greedy, beam-search, ...

Masked Self-Attention: Motivation

In language modeling, we want to predict the next token given the previous ones:

- ▶ Given a sentence from the training data: “The cat sat on the mat” we want to predict the next word for each word in the sentence:
 - $x = \text{“The”} ; y = \text{“cat”}$
 - $x = \text{“The cat”} ; y = \text{“sat”}$
 - ...
 - $x = \text{“The cat sat on the”} ; y = \text{“mat”}$
- ▶ We can train on all these examples simultaneously.

Masked Self-Attention: Motivation

Attention weight between the tokens corresponding to “Life” and “short”

	Life	is	short	eat	desert	first
Life	0.17	0.13	0.18	0.16	0.15	0.18
is	0.03	0.68	0.02	0.08	0.14	0.02
short	0.19	0.06	0.25	0.14	0.11	0.23
eat	0.15	0.21	0.14	0.16	0.17	0.14
desert	0.13	0.27	0.11	0.16	0.18	0.12
first	0.19	0.02	0.31	0.11	0.07	0.27

Masked out future tokens are grayed out

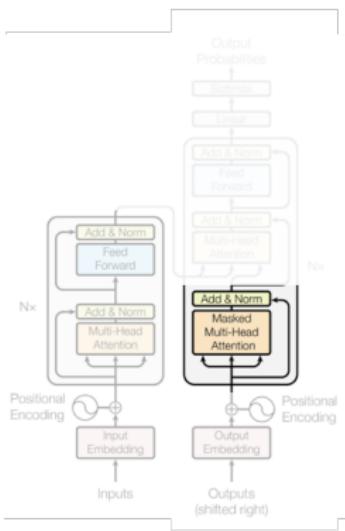


	Life	is	short	eat	desert	first
Life	0.17					
is	0.03	0.68				
short	0.19	0.06	0.25			
eat	0.15	0.21	0.14	0.16		
desert	0.13	0.27	0.11	0.16	0.18	
first	0.19	0.02	0.31	0.11	0.07	0.27

Masked self-attention

At training time: Masked self-attention

- ▶ If we had to train on one single $p(z_3 | z_2, z_1, x)$ at a time: SLOW! Instead, train on all $p(z_i | z_{1:i}, x)$ simultaneously.
- ▶ How? Mask out the future tokens in the input sequence. In the attention weights for z_i , set all entries $i : N$ to 0 . Each token only sees the already generated ones.

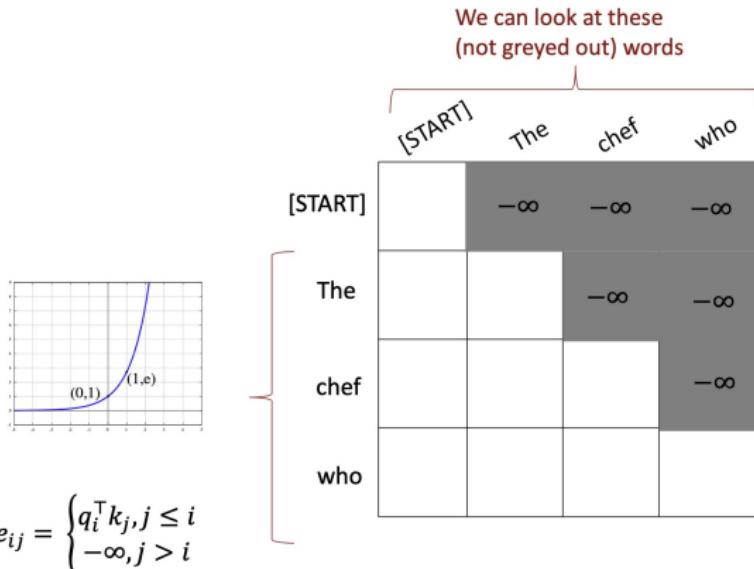


At generation time:

- ▶ There is no such trick. Must generate one z_i at a time. Hence, autoregressive decoding is really slow.

Attention Masking Details

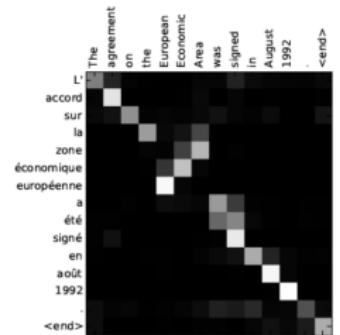
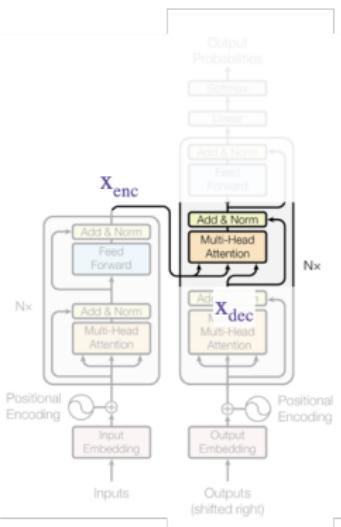
- ▶ We want to mask with upper triangular matrix. In PyTorch use: `torch.triu`
- ▶ Setting **attention score** to $-\infty$ ensures that attention probability will be zero.



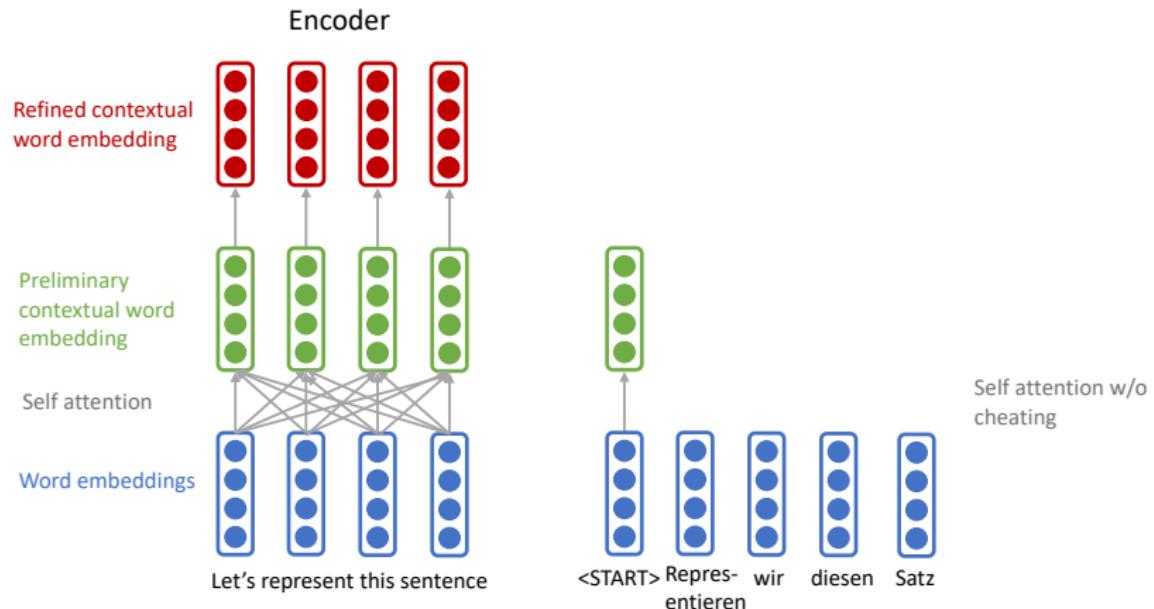
Encoder-Decoder Attention (Cross Attention)

Encoder-Decoder Attention

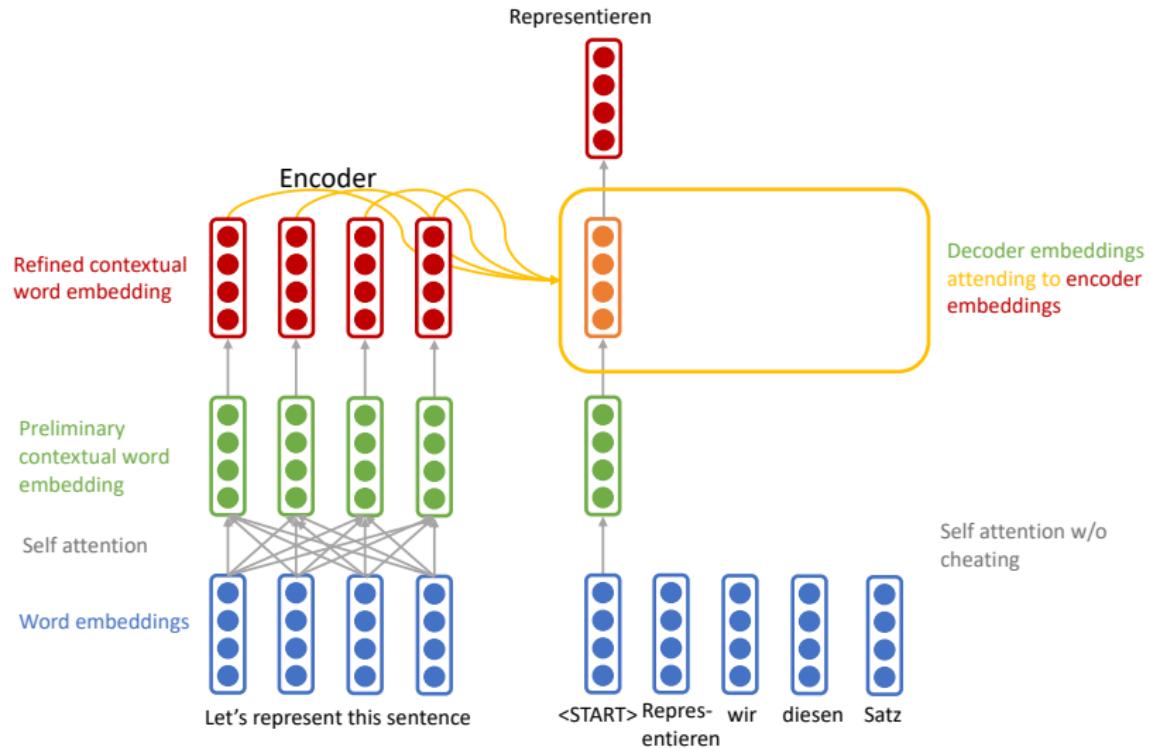
- ▶ Each decoded token can "look at" the encoder's output (as in seq2seq w/ attention)
- ▶ This allows the decoder to focus on relevant parts of the encoder's output.



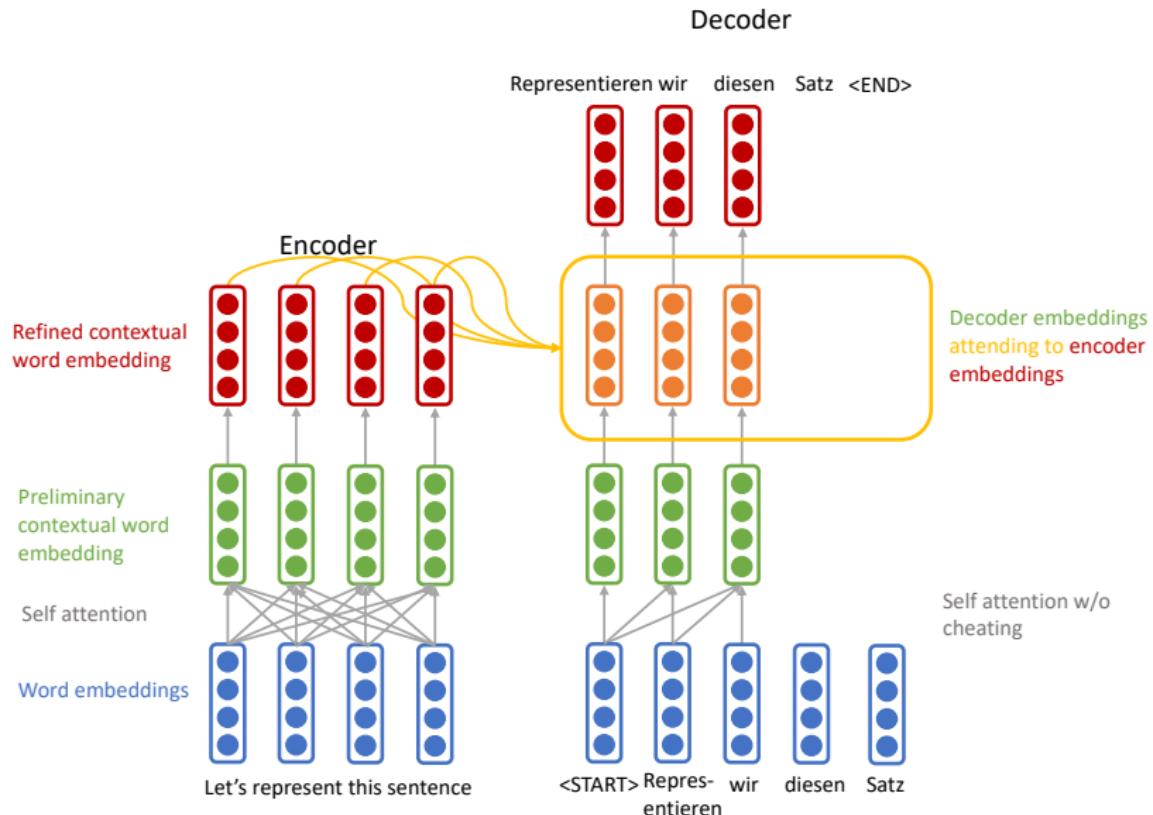
Transformer Encoder-Decoder



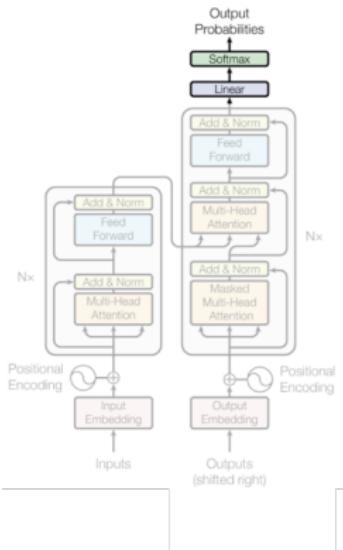
Transformer Encoder-Decoder



Transformer Encoder-Decoder



Output Layer

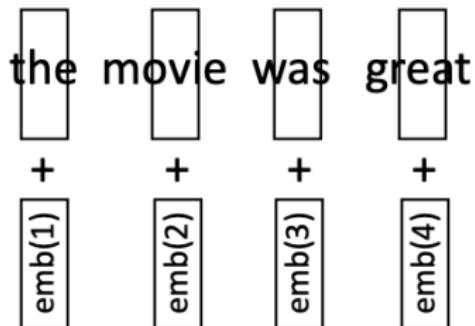
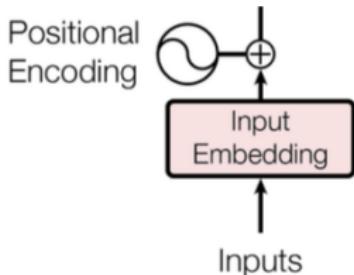


- ▶ The output layer converts the decoder output into a probability distribution over the vocabulary.
 - ⦿ The output layer is a linear layer with a softmax activation.
- ▶ (Greedy decoding): The word with the highest probability is the predicted next word in the output sequence.

Positional Encoding

Position Sensitivity

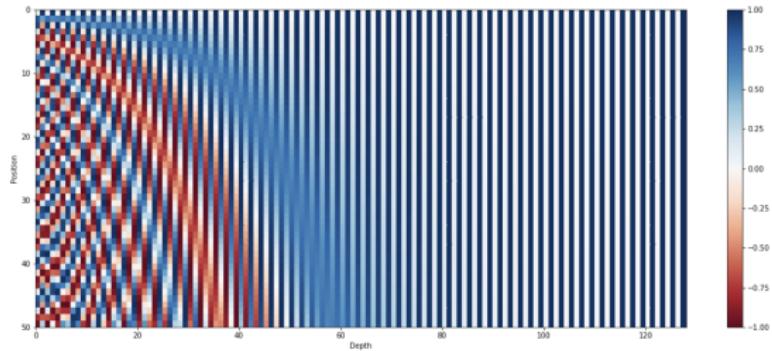
"the cat sat on the mat"



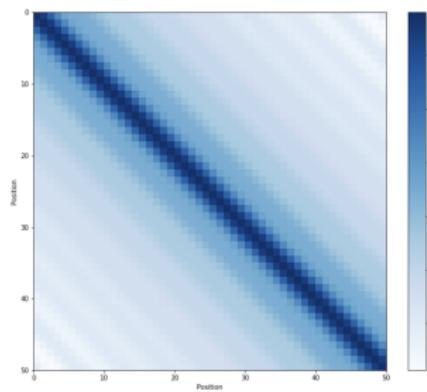
- ▶ Encode each sequence position as an integer, add its embedding to the word embedding vector
- ▶ Separate embedding matrix from the word embeddings

Using Trigonometric Functions for Positional Encoding

- ▶ Vaswani et al 2017: Calculate each dimension with sines/cosines of different frequencies



Why? closer words get higher dot products



Absolute vs Relative Positional Encodings

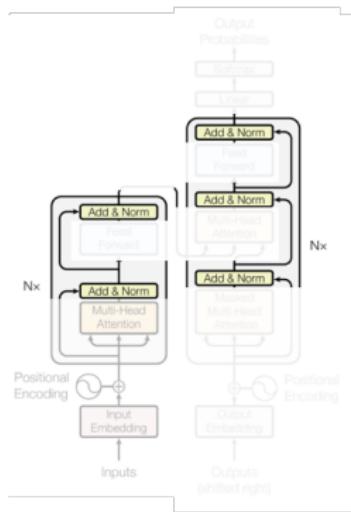
- ▶ **Absolute:** add an encoding to the input in **hope** that relative position will be captured
 - Vaswani et al. 2017: Positional Encodings
 - Fails to extrapolate to longer sequences
- ▶ **Relative:** **Explicitly** encode relative position e.g., "I" is 3 words away from "run"
 - Press et al. 2021: "Attention with Linear Biases" (ALiBi), **parameter-free** linear bias added to dot-product similarity. Extrapolates to longer sequences.

$$\begin{array}{c} \begin{matrix} q_1 \cdot k_1 \\ q_2 \cdot k_1 \quad q_2 \cdot k_2 \\ q_3 \cdot k_1 \quad q_3 \cdot k_2 \quad q_3 \cdot k_3 \\ q_4 \cdot k_1 \quad q_4 \cdot k_2 \quad q_4 \cdot k_3 \quad q_4 \cdot k_4 \\ q_5 \cdot k_1 \quad q_5 \cdot k_2 \quad q_5 \cdot k_3 \quad q_5 \cdot k_4 \quad q_5 \cdot k_5 \end{matrix} + \begin{matrix} 0 \\ -1 \quad 0 \\ -2 \quad -1 \quad 0 \\ -3 \quad -2 \quad -1 \quad 0 \\ -4 \quad -3 \quad -2 \quad -1 \quad 0 \end{matrix} \bullet m \end{array}$$

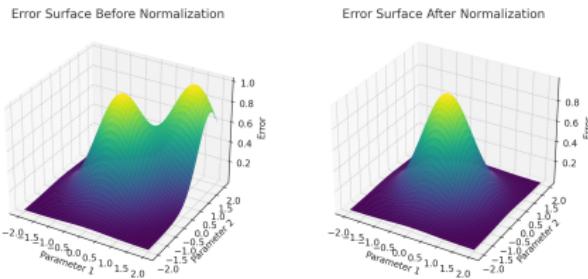
Training Transformers - Optimization Tricks

Residual Connections and Layer Normalization

Residual connections: training converges faster, and allows for deeper networks.



Layer normalization: Normalizes the outputs to be within a consistent range, preventing too much variance in scale of outputs



Transformer Summary and Results

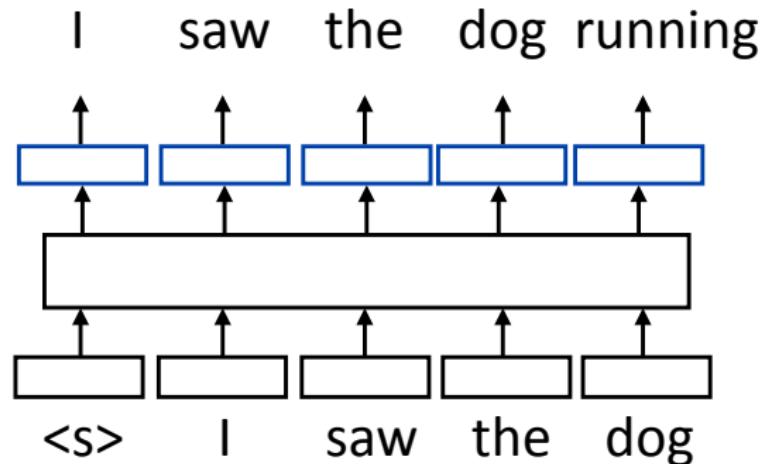
Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

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	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	$2.3 \cdot 10^{19}$	

10^{21} is 100 times larger than 10^{19} . If something takes 40 days in the 10^{21} regime, it would take 0.4 days (or about 9.6 hours) in the 10^{19} regime.

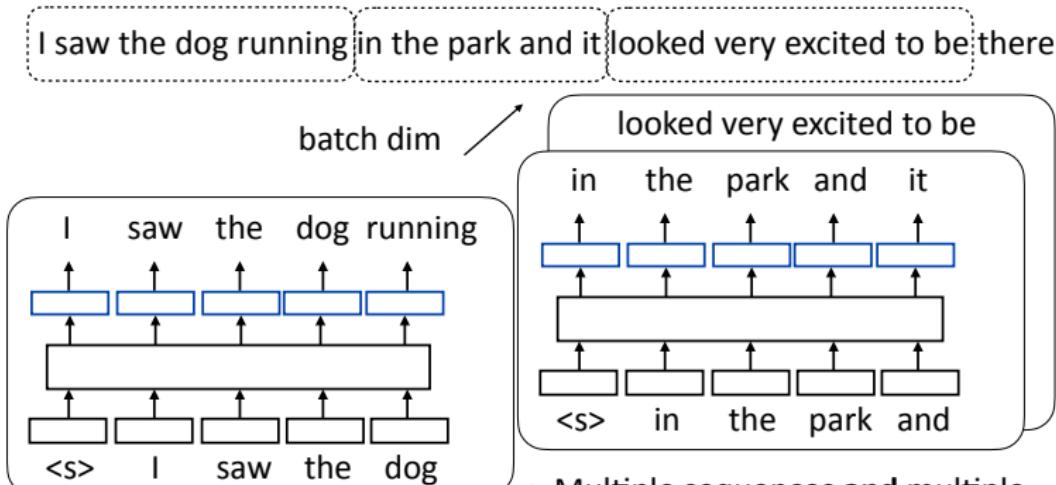
Demo

Transformer Language Modeling: Training



- ▶ Input is a sequence of words, output is those words shifted by one,
- ▶ Allows us to train on predictions across several timesteps simultaneously (similar to batching but this is NOT what we refer to as batching)

Batched LM Training



- ▶ Multiple sequences **and** multiple timesteps per sequence

What happens with Multi-Head Attention?

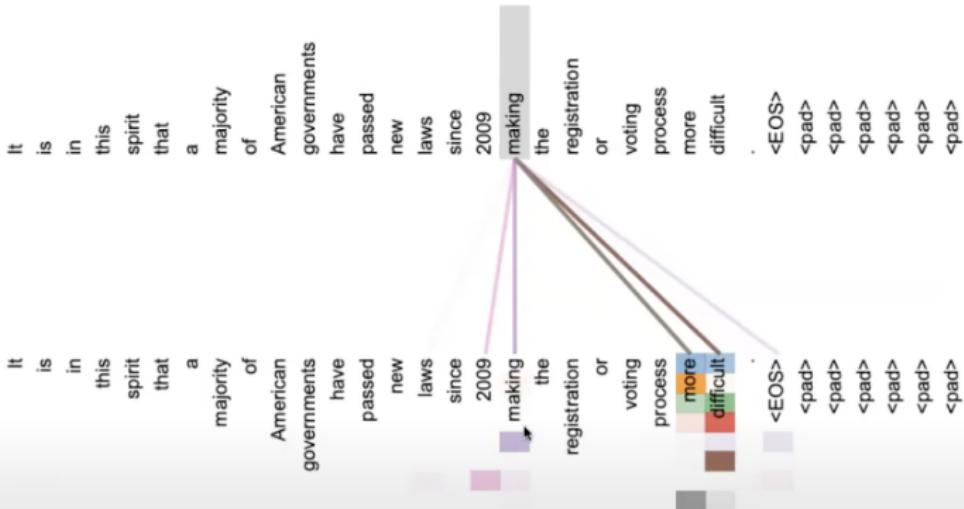


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word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

Sequence Modeling

Challenges with RNNs

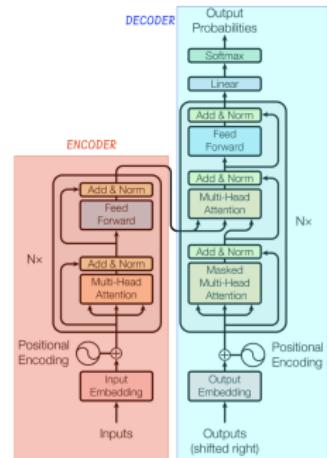
- ▶ Gradient vanishing and explosion
 - Long-range dependencies are not captured
- ▶ Recurrence prevents parallel computation
 - Slow training

Transformer Networks

- ▶ Facilitate long range dependencies
- ▶ No gradient vanishing and explosion
- ▶ No recurrence: facilitate parallel computation

The Transformer: versatile architecture

Domain	Data Type
Natural Languages	Text
Programming Languages	Code
Computer Vision	Images
Speech	Audio
Proteins Folding	Protein Sequences
Music	Audio
Video	Video
...	...



Takeaways

- ▶ Transformers are going to be the foundation for the much of the rest of this class and are a ubiquitous architecture nowadays
- ▶ Many details to get right, many ways to tweak and extend them, but core idea is the multi-head self attention and their ability to contextualize items in sequences

Transformer Runtime

- ▶ Even though most parameters and FLOPs are in feedforward layers, Transformers are still limited by quadratic $O(n^2)$ complexity of self-attention
- ▶ Quadratic complexity, but $O(1)$ sequential operations (not linear like in RNNs) and $O(1)$ "path" for words to inform each other ... so it's still fast
- ▶ Many tricks to reduce this complexity, e.g., Longformer, Reformer, Linformer, ...

Longformer

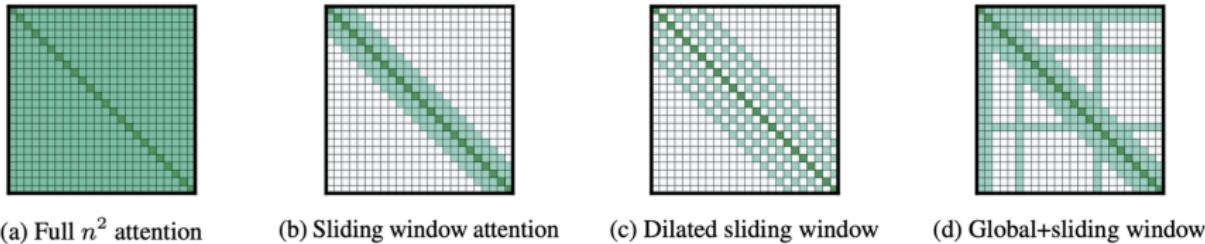


Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer

- ▶ Use several pre-specified self-attention patterns that limit the number of operations
- ▶ Scales to 4096 -length sequences
- ▶ Engineering-based approaches like Flash Attention (which supports the "basic" Transformer) have superseded changing the Transformer model itself

That's it!

Acknowledgements: Includes content from L. Beyer, and others
Any omissions in the attribution will be corrected upon notification.