

Redes Neurais/Cibernética/Ap.Máquina

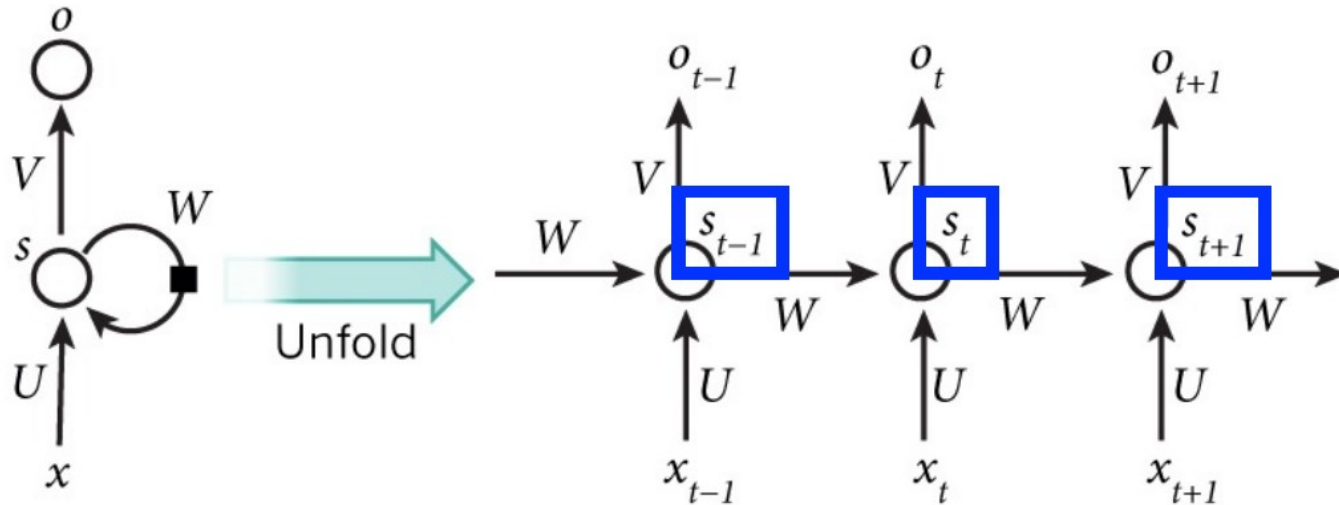
- Autoatenção e *Transformer*
-
- Exemplos



Imagem: pixabay.com

RNNs

Problem: RNNs Use Sequential Computation

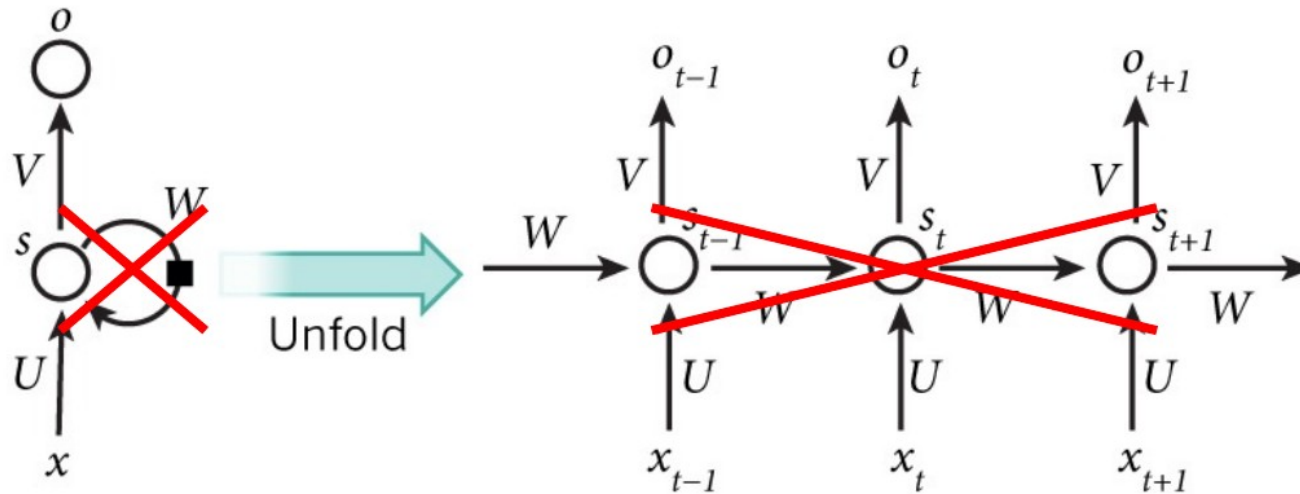


Seemingly hard for RNNs to carry information through hidden states across many time steps and train/testing is slow

<http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/>
(slides/images D. Gurari)

RNNs e Autoatenção

Idea: Model Sequential Data Without Recurrence



Replace sequential hidden states for capturing knowledge of other inputs with a new representation of each input that shows its relationship to all other inputs (i.e., self-attention)

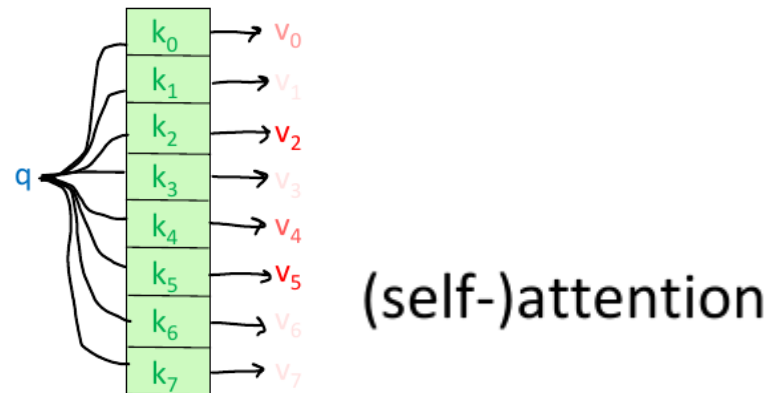
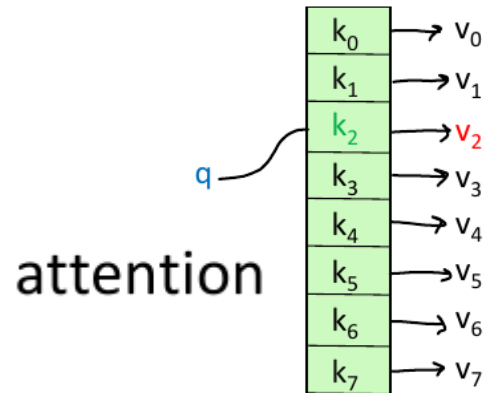
<http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/>

(slides/images D. Gurari)

Autoatenção?

Intuition for Attention Mechanism

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



(slides/images C. Manning)

Autoatenção?

Transformer vs RNN (Intuition)

I arrived at the **bank** after crossing the...

...street? ...river?

What does **bank** mean in this sentence? Meaning depends on other input words



I've no idea: let's wait until I read the end

RNNs

$O(N)$ steps to process a sentence with length N



I don't need to wait - I see all words at once!

Transformer

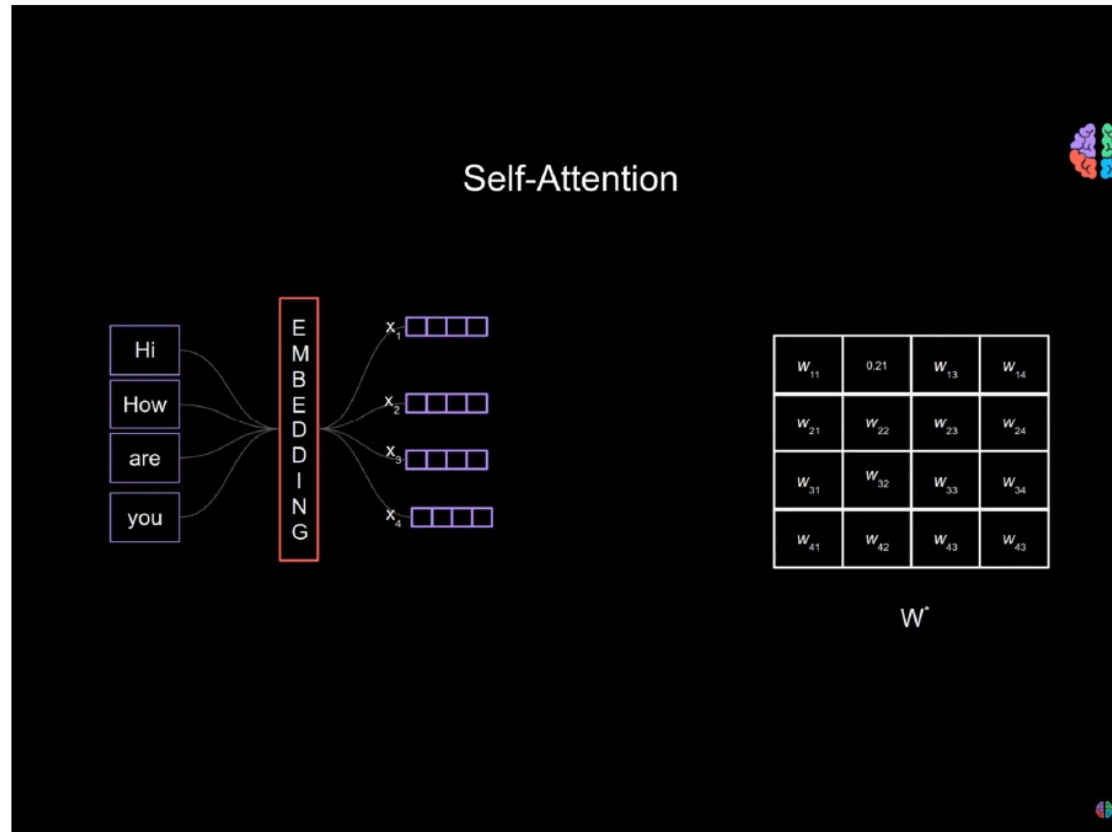
Constant number of steps to process any sentence

https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html#transformer_intro

(slides/images D. Gurari)

Autoatenção?

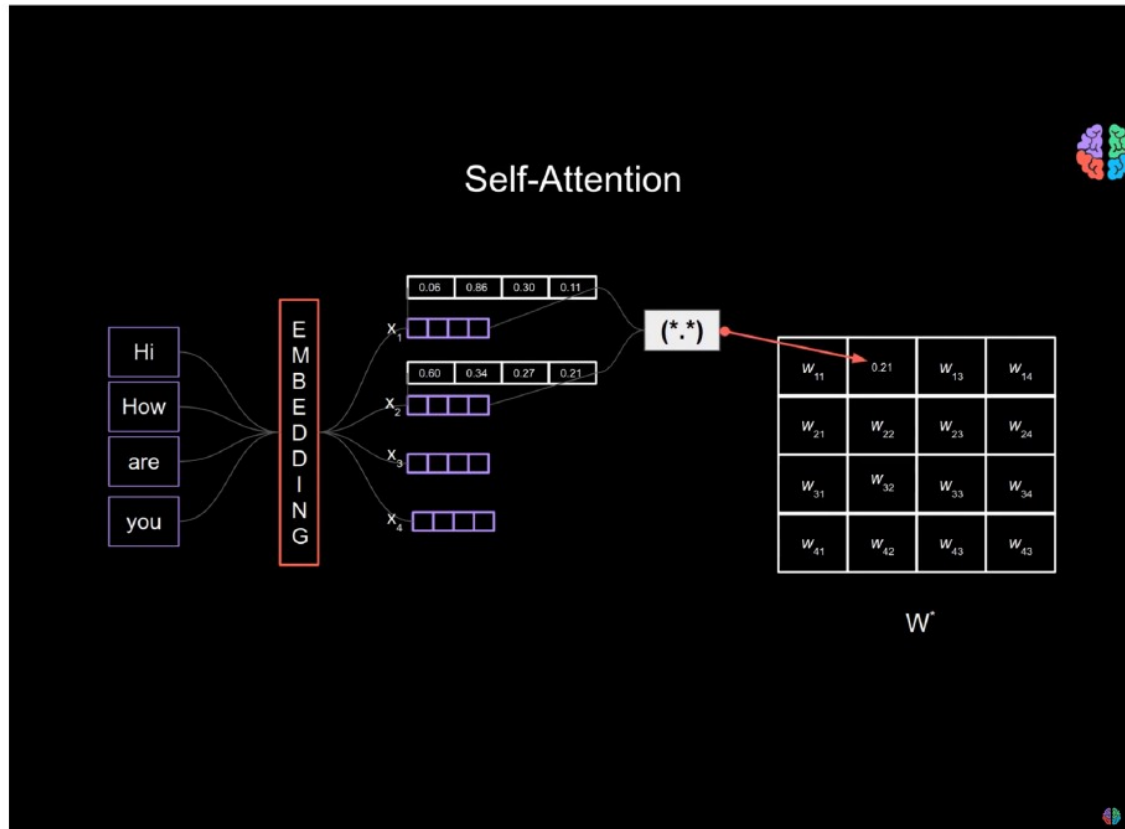
Slide courtesy of AI Bites, Youtube Channel:
<https://www.youtube.com/c/AIBites>



1/11/2024

(slides/images M. Shah)

Autoatenção?

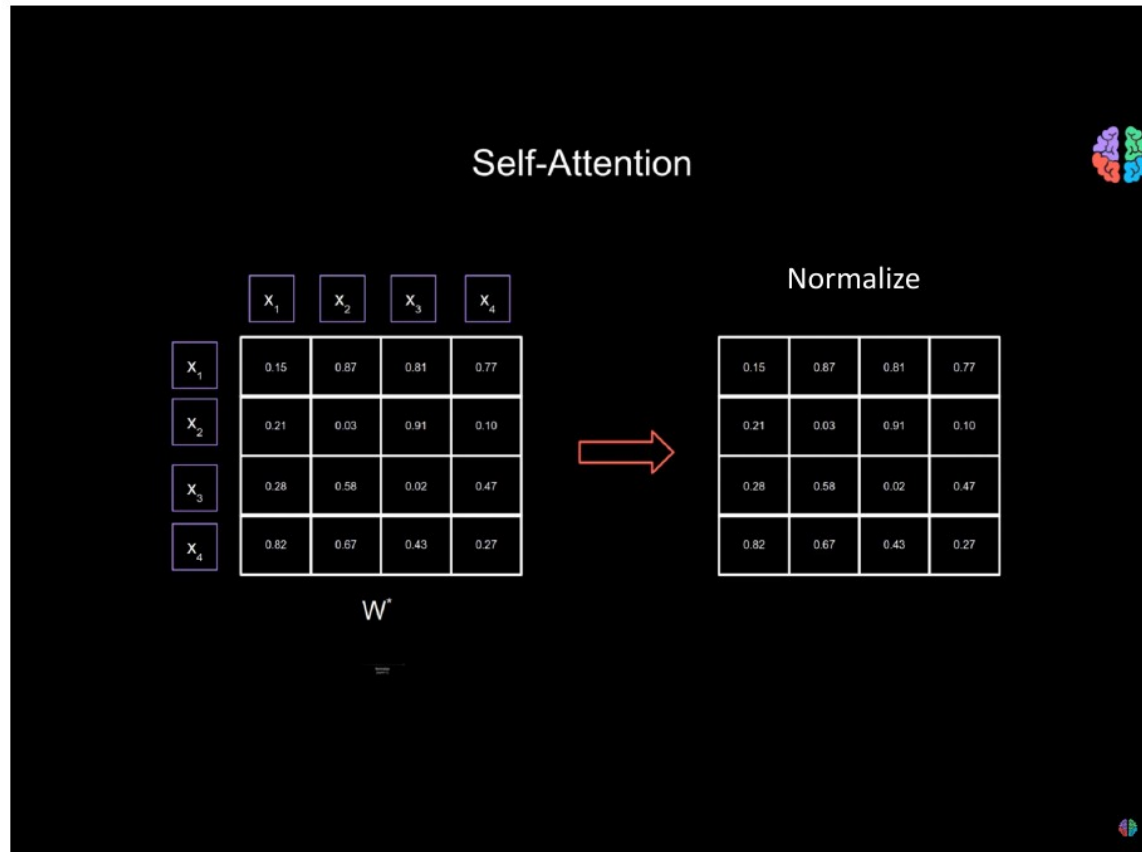


Slide courtesy of AI Bites, Youtube Channel:
<https://www.youtube.com/c/AIBites>

1/11/2024

(slides/images M. Shah)

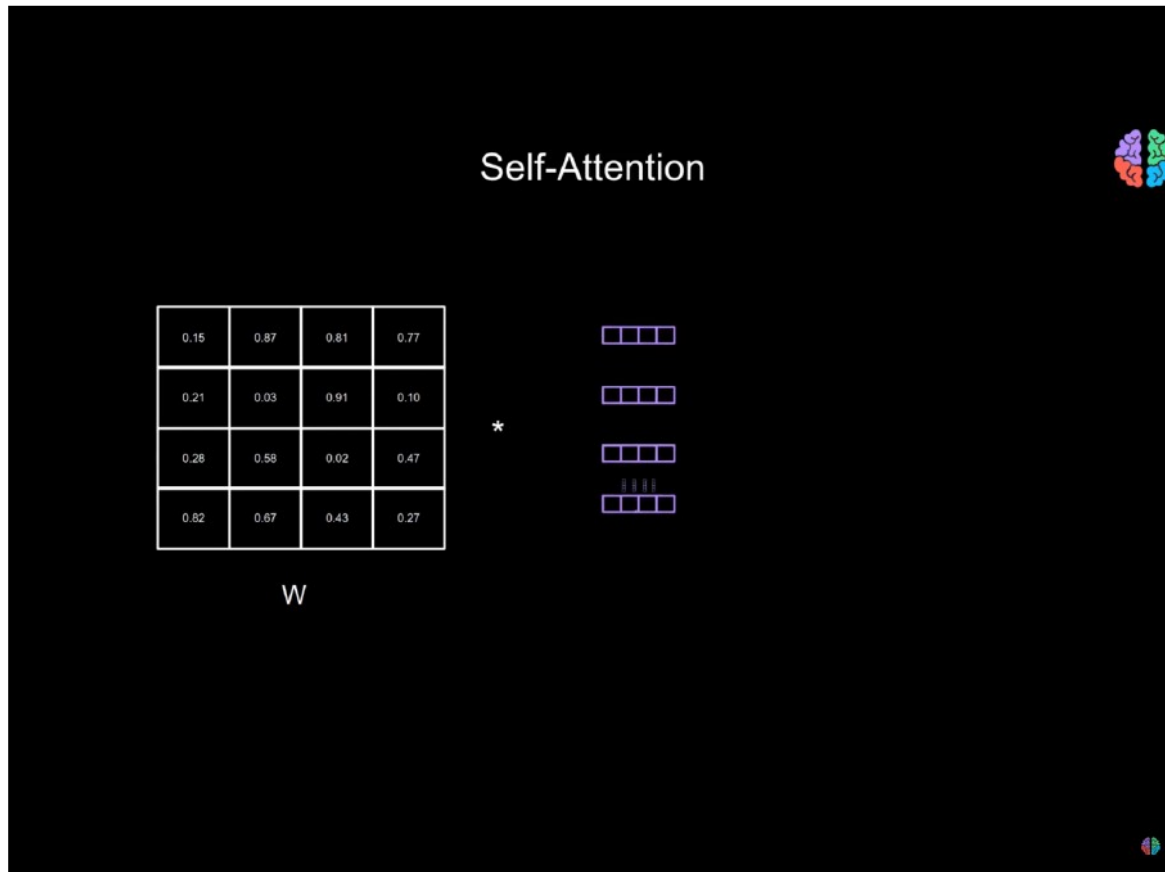
Autoatenção?



Slide courtesy of AI Bites, Youtube
Channel:
<https://www.youtube.com/c/AIBites>

(slides/images M. Shah)

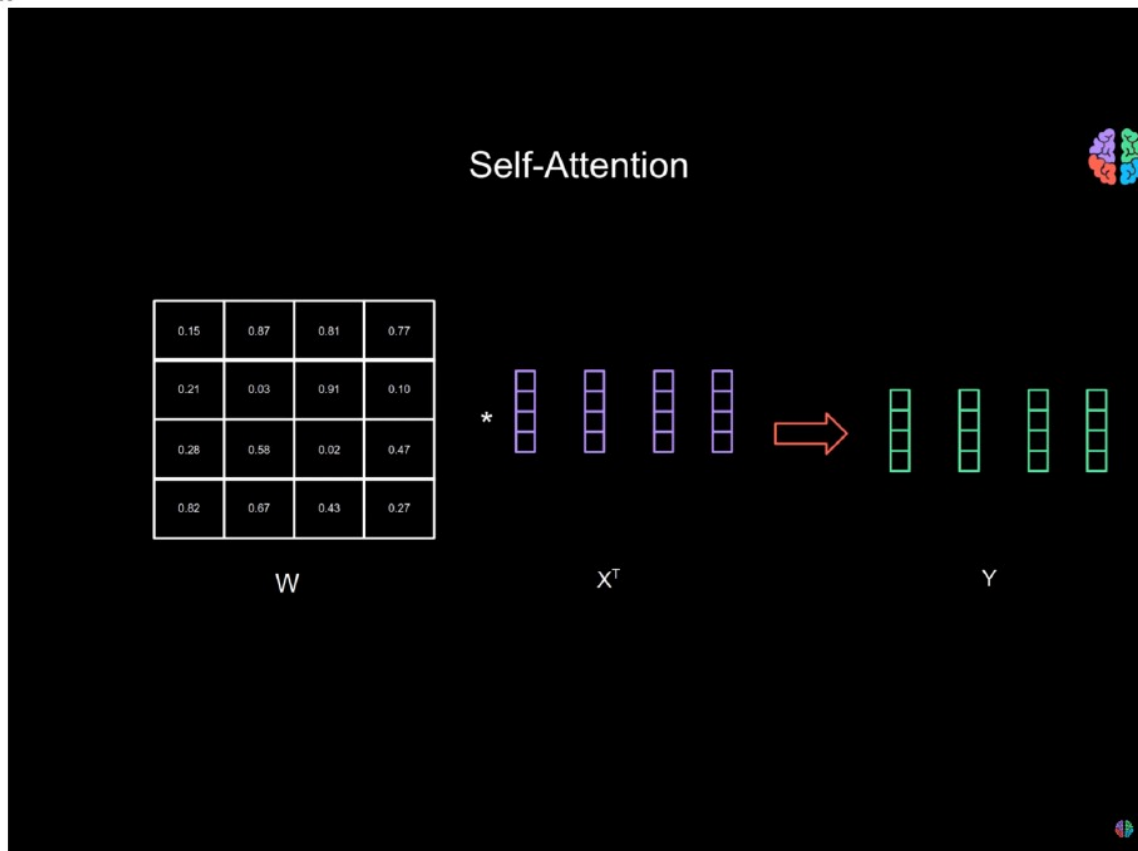
Autoatenção?



Slide courtesy of AI Bites, Youtube Channel:
<https://www.youtube.com/c/AIBites>

(slides/images M. Shah)

Autoatenção?



Slide courtesy of AI Bites, Youtube
Channel:
<https://www.youtube.com/c/AIBites>

(slides/images M. Shah)

Autoatenção?

Self-Attention

Lets denote a sequence of n entities ($\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$)

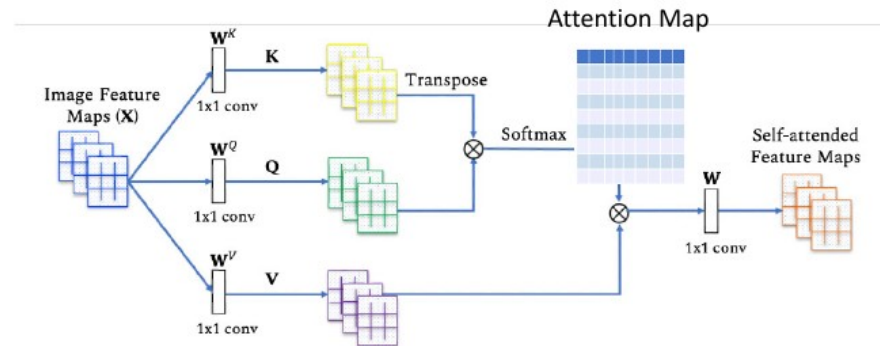
by $\hat{\mathbf{X}} \in \mathbb{R}^{n \times d}$
 9×3

Queries ($\mathbf{W}^Q \in \mathbb{R}^{d \times d_q}$, 3×3), Keys ($\mathbf{W}^K \in \mathbb{R}^{d \times d_k}$, 3×3) and Values ($\mathbf{W}^V \in \mathbb{R}^{d \times d_v}$, 3×3), where $d_q = d_k$. The input sequence \mathbf{X} is

first projected onto these weight matrices to get $\mathbf{Q} = \mathbf{X}\mathbf{W}^Q$, 9×3 9×3 3×3
 $\mathbf{K} = \mathbf{X}\mathbf{W}^K$ and $\mathbf{V} = \mathbf{X}\mathbf{W}^V$. The output $\mathbf{Z} \in \mathbb{R}^{n \times d_v}$ of the
 9×3 9×3

$$\mathbf{Z} = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_q}} \right) \mathbf{V}.$$

9×3 9×9 9×3



(slides/images M. Shah)

Transformer

Transformer: A Suggested Definition

“Any architecture designed to process a connected set of units—such as the tokens in a sequence or the pixels in an image—where the only interaction between units is through self-attention.”

<http://peterbloem.nl/blog/transformers>

(slides/images D. Gurari)

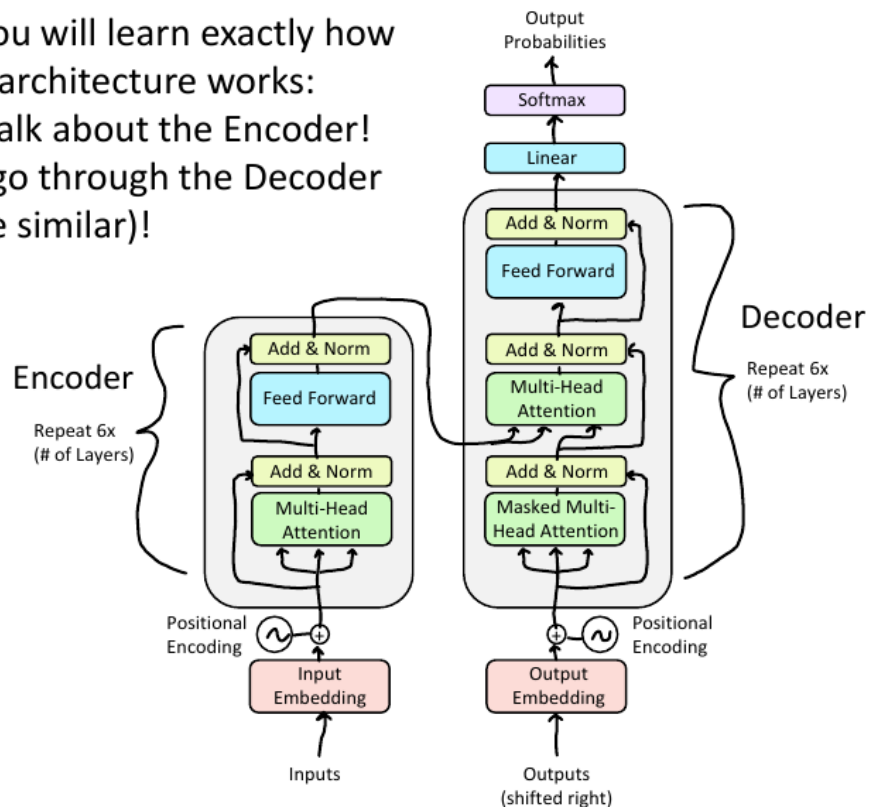


Transformer

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

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

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

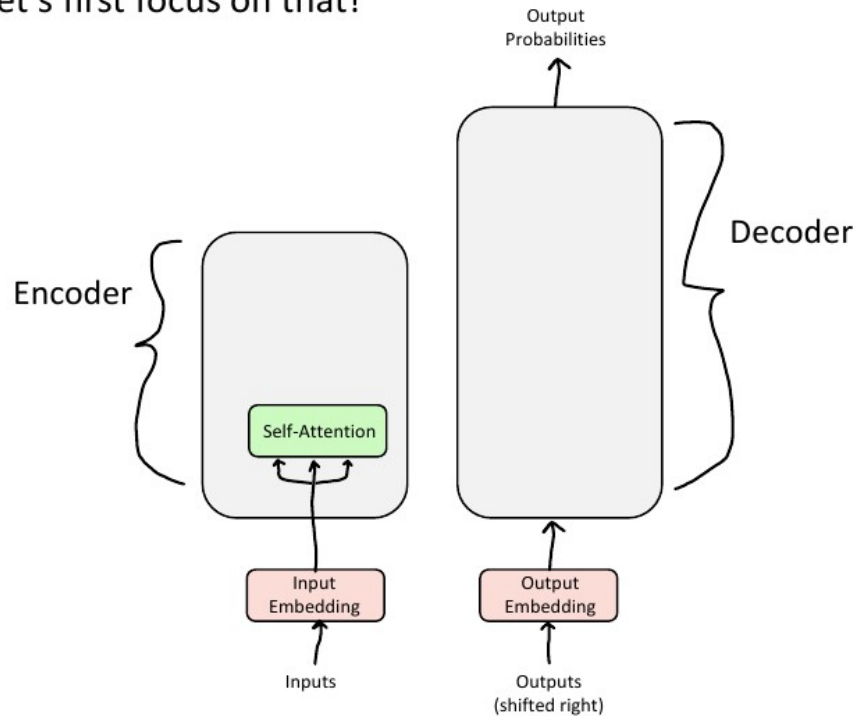


(slides/images C. Manning)

Transformer

Encoder: Self-Attention

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



(slides/images C. Manning)

Transformer

Recipe for Self-Attention in the Transformer Encoder

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

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

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

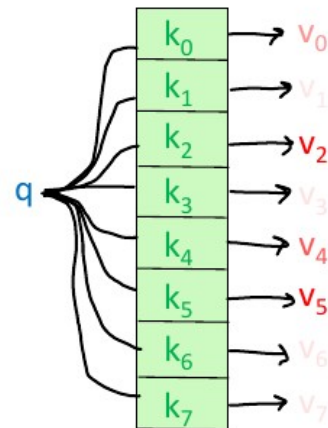
$$e_{ij} = q_i \cdot k_j$$

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

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

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

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



(slides/images C. Manning)

Transformer

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

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

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

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

$$E = QK^T$$

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

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

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

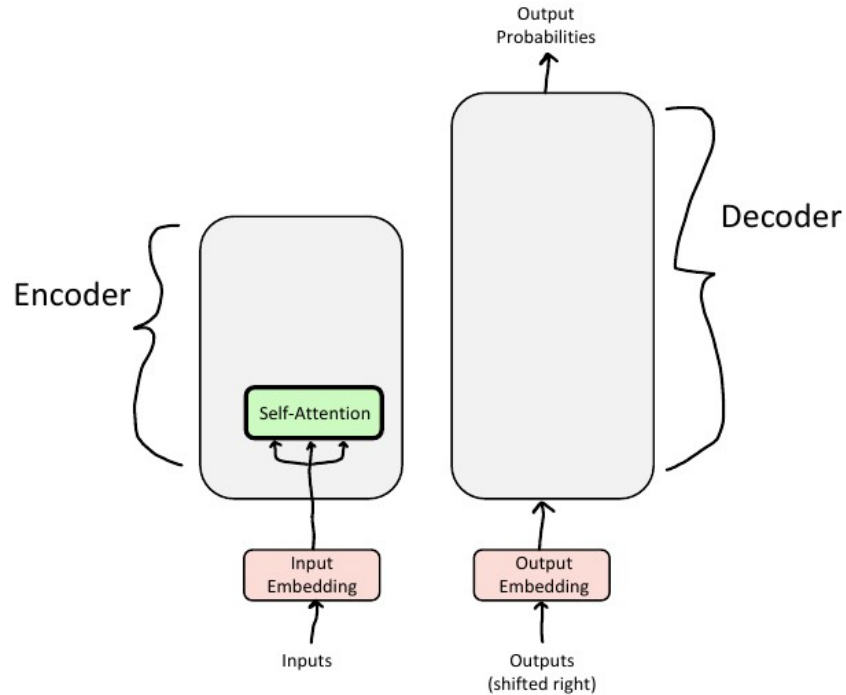
$$\text{Output} = AV$$

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

(slides/images C. Manning)

Transformer

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

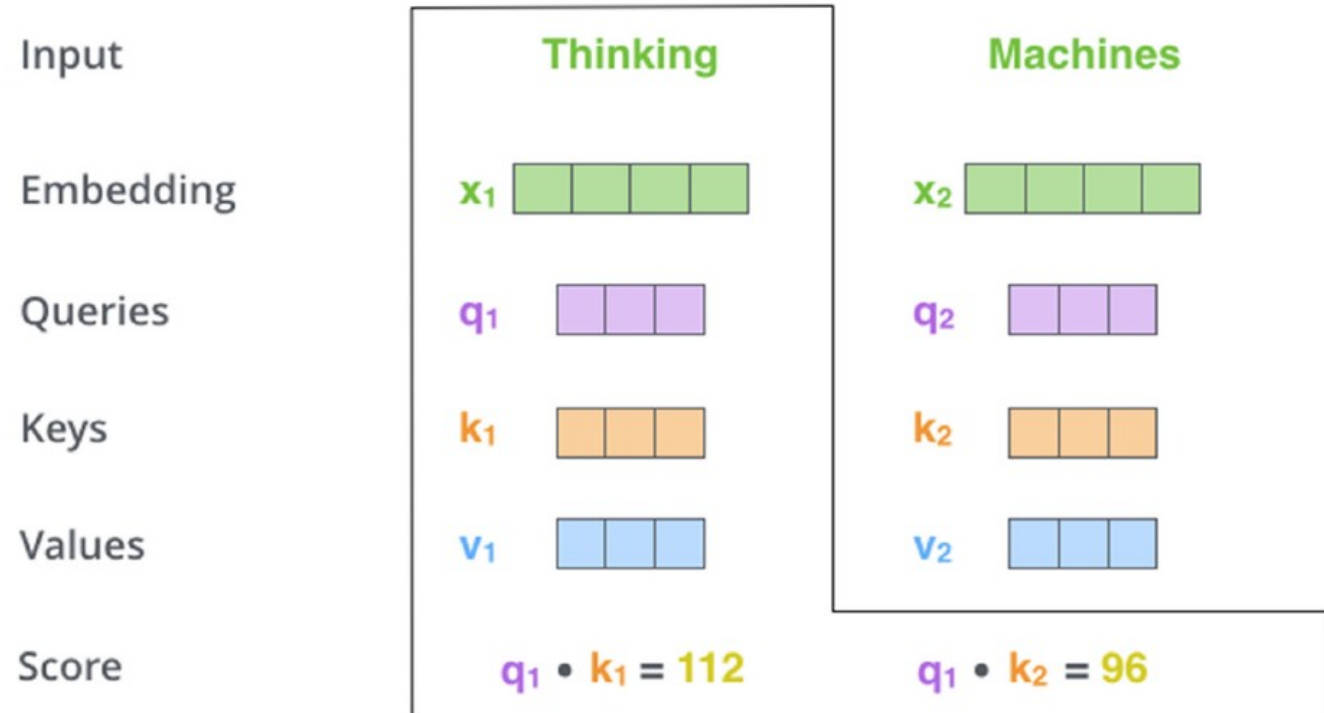


(slides/images C. Manning)

Transformer

Self Attention

Now we calculate a score to determine how much focus to place on other Parts of the input.



Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP

Transformer



Attention and Transformer

Self Attention

Input

Embedding

Queries

Keys

Values

Score

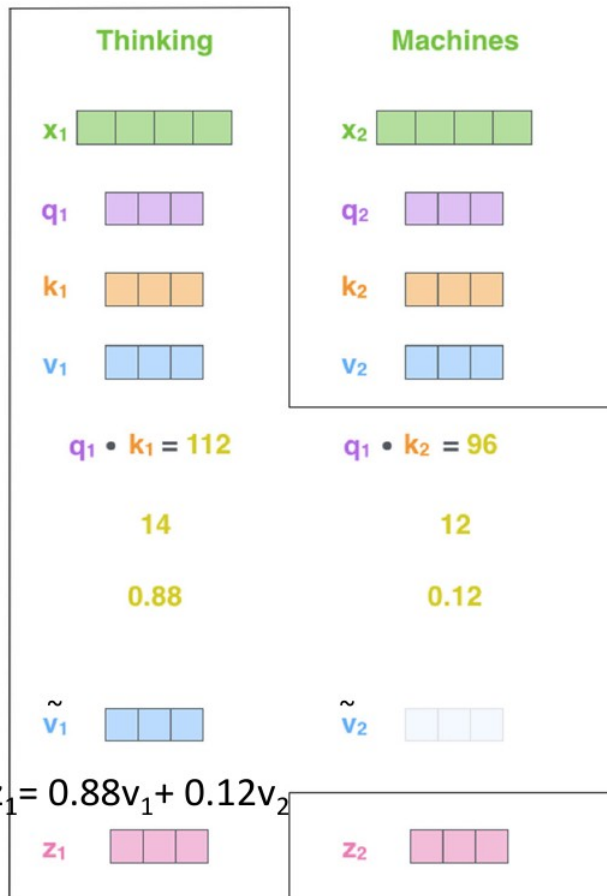
Divide by $8 (\sqrt{d_k})$

Softmax

Softmax

X
Value

Sum



Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP

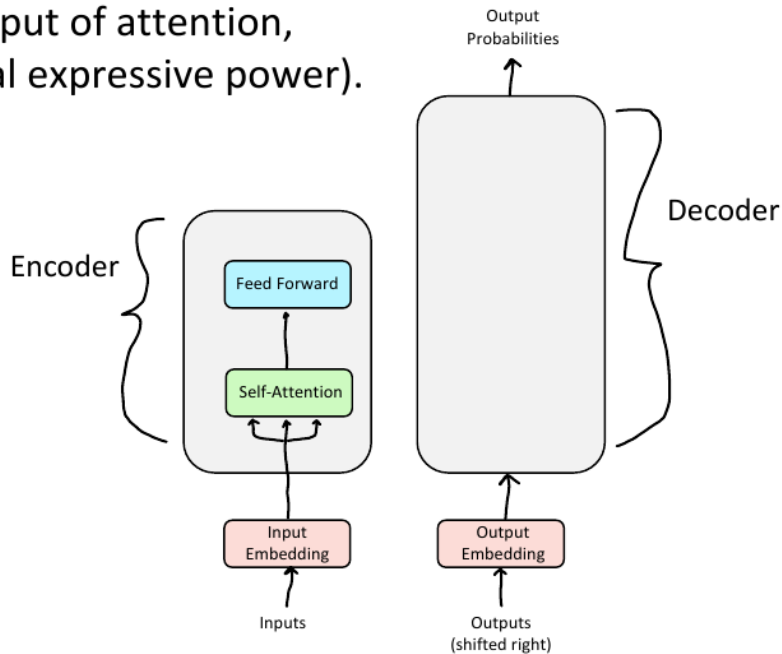
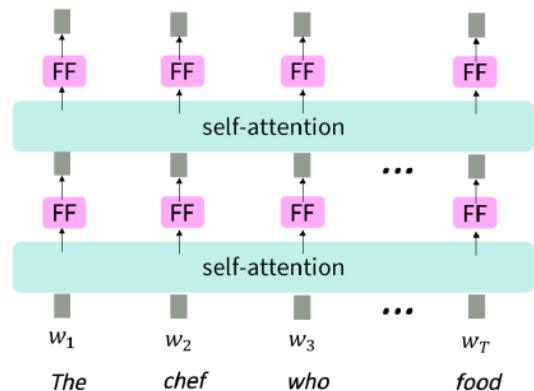
Transformer

But attention isn't quite all you need!

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

Equation for Feed Forward Layer

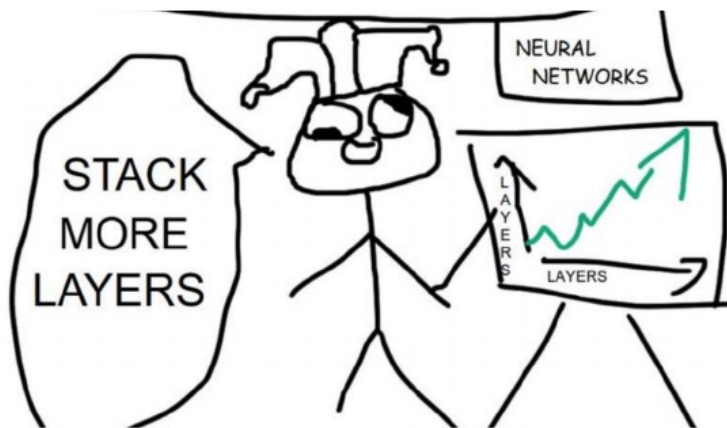
$$m_i = MLP(\text{output}_i) \\ = W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$



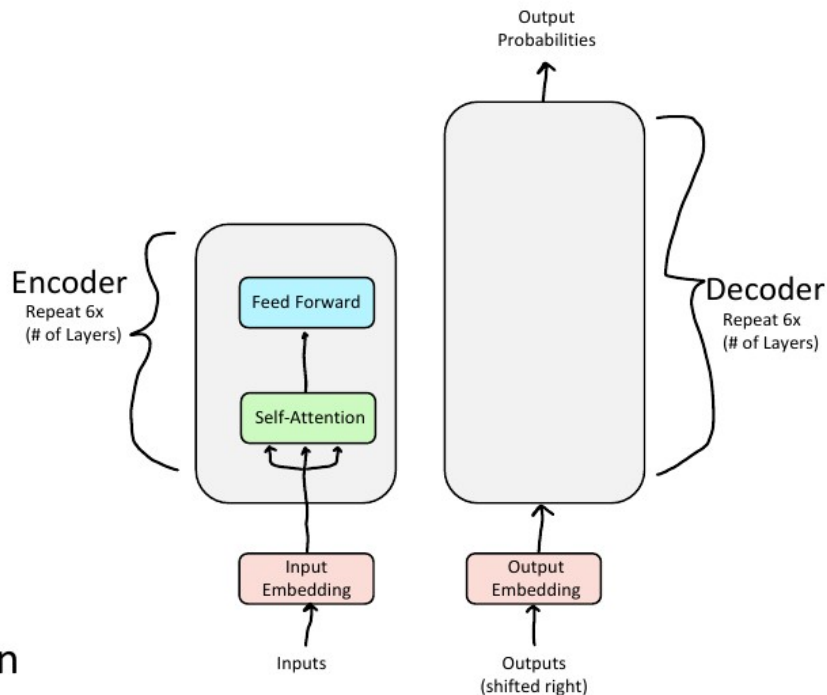
(slides/images C. Manning)

Transformer

But how do we make this work for deep networks?



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

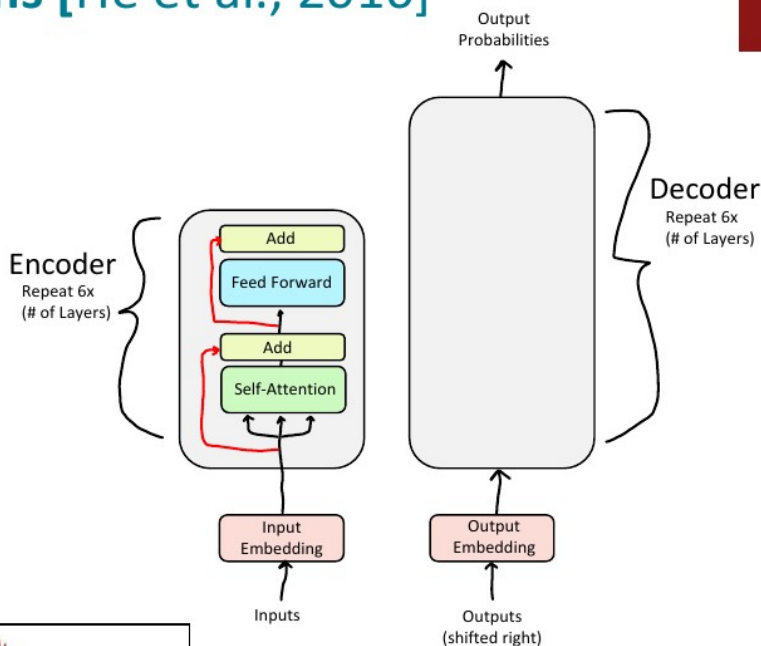


(slides/images C. Manning)

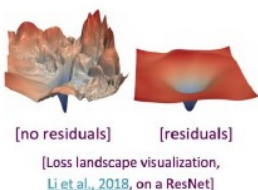
Transformer

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!



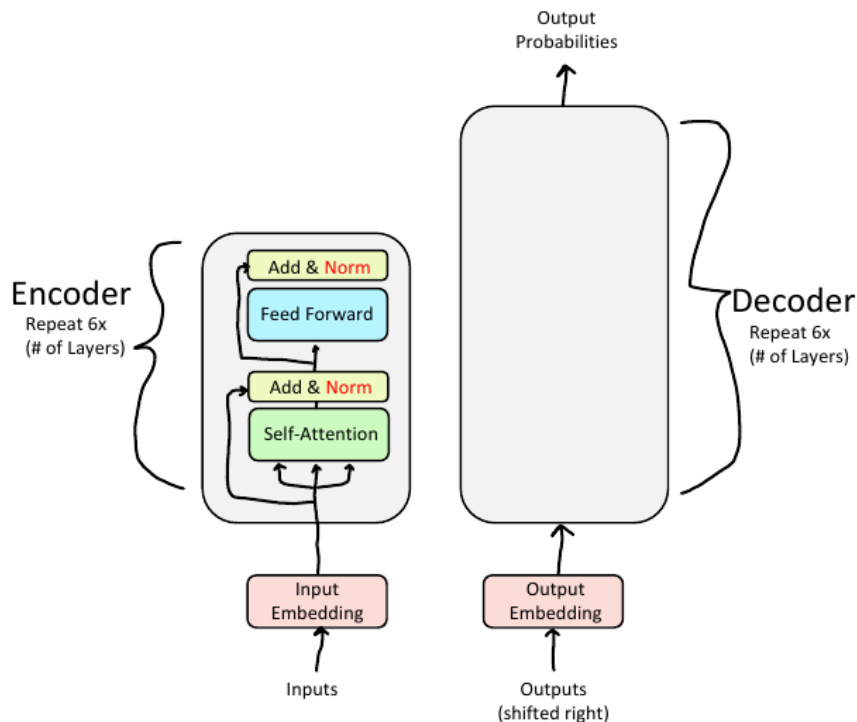
Transformer

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

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

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

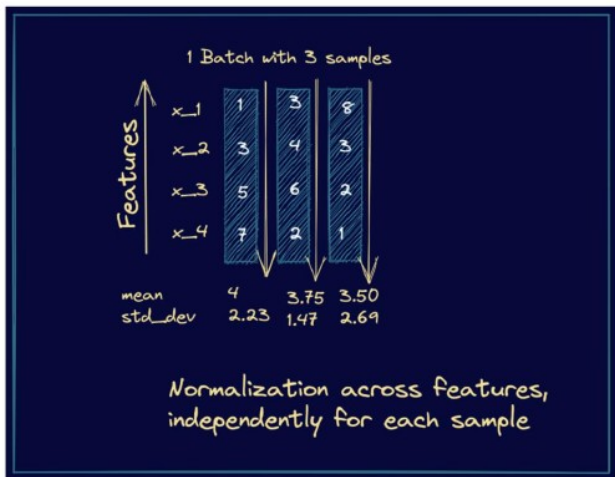
$$x^{\ell'} = \frac{x^{\ell} - \mu^{\ell}}{\sigma^{\ell} + \epsilon}$$



(slides/images C. Manning)

Transformer

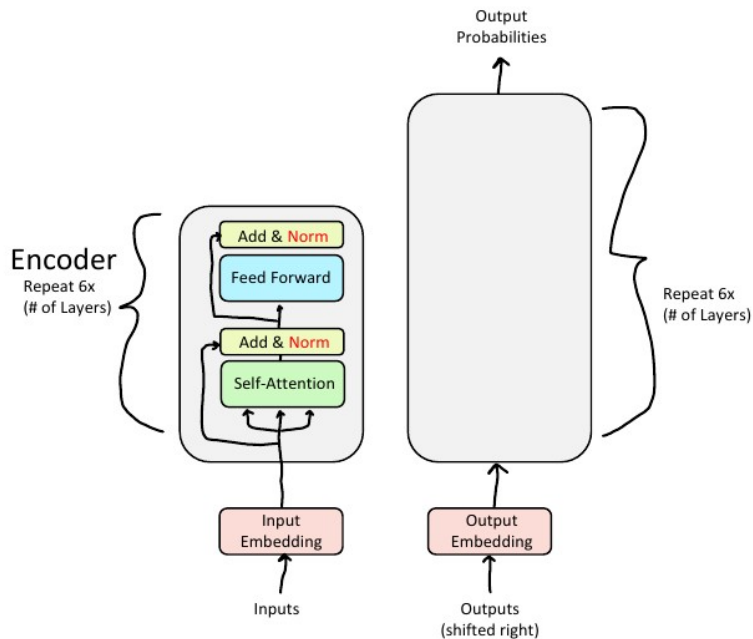
Training Trick #2: Layer Normalization [Ba et al., 2016]



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

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

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



(slides/images C. Manning)

Transformer

Training Trick #3: Scaled Dot Product Attention

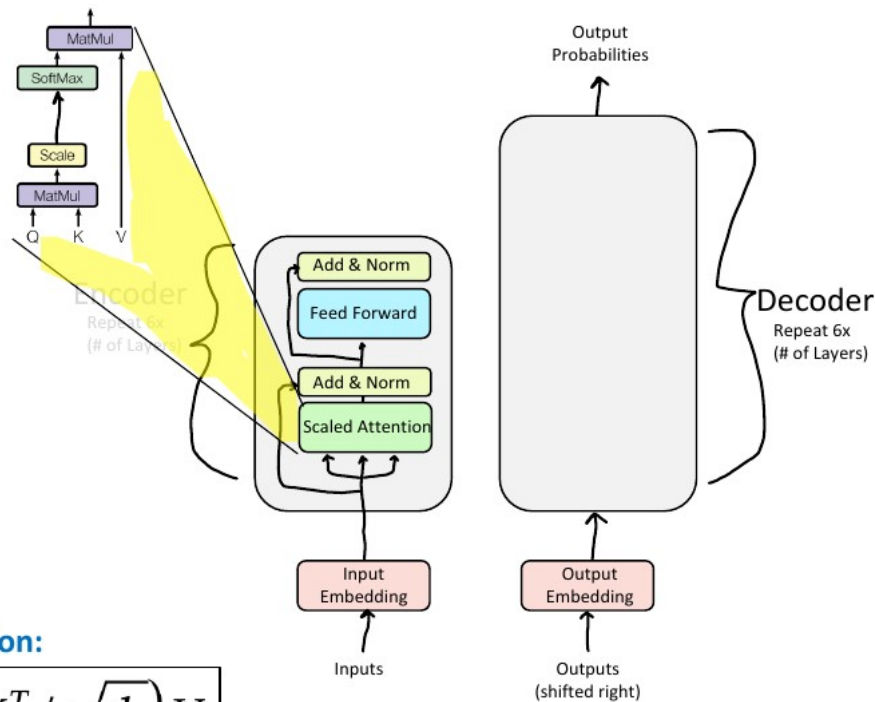
- After LayerNorm, the mean and variance of vector elements is 0 and 1, respectively. (Yay!)
- However, the dot product still tends to take on extreme values, as its variance scales with dimensionality d_k

Quick Statistics Review:

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

Updated Self-Attention Equation:

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

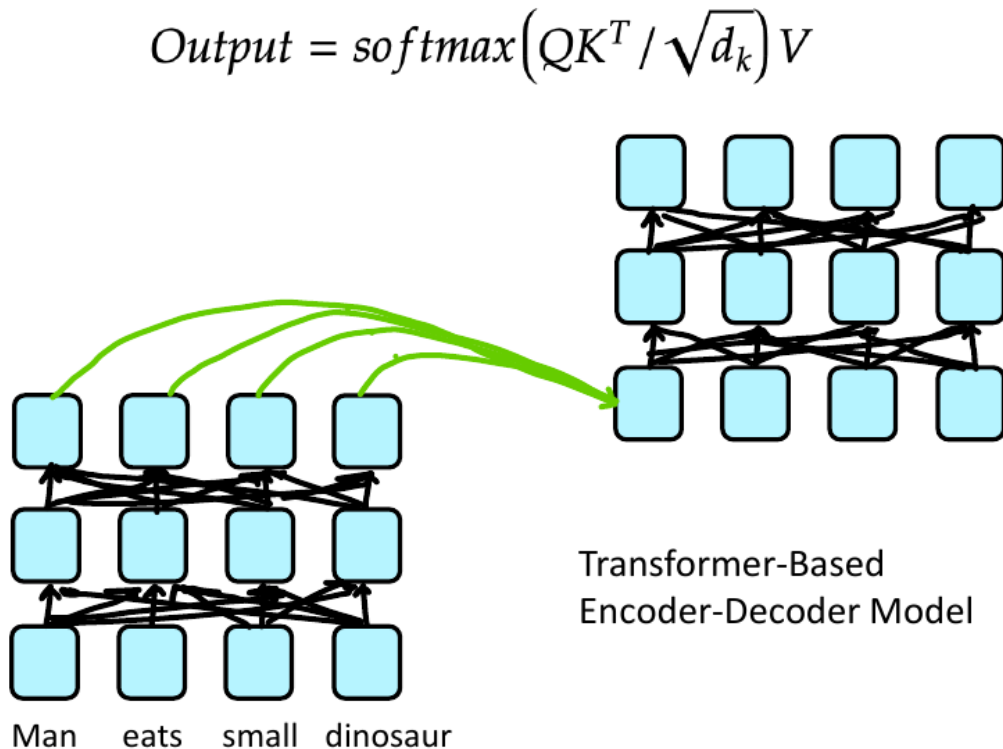


(slides/images C. Manning)

Transformer

Major issue!

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

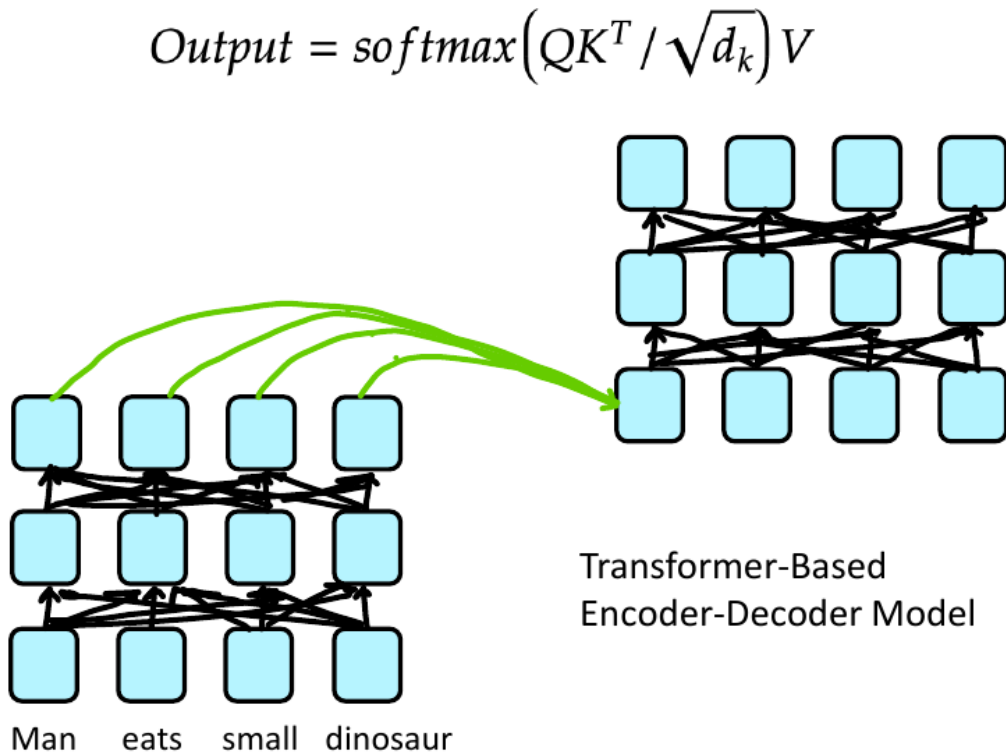


(slides/images C. Manning)

Transformer

Major issue!

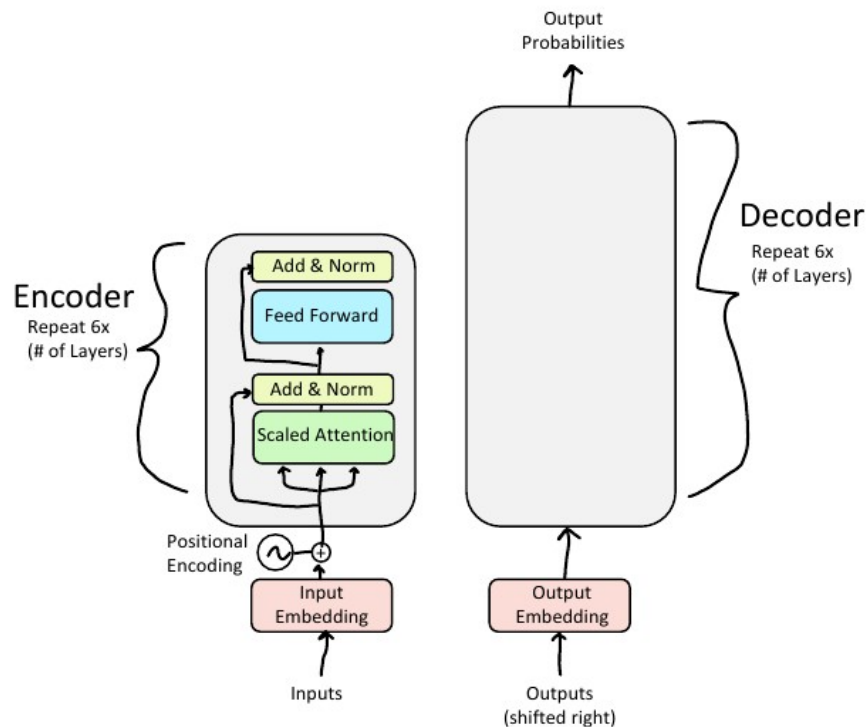
- We're almost done with the Encoder, but we have a major problem! Has anyone spotted it?
- Consider this sentence:
 - "Man eats small dinosaur."
- Wait a minute, order doesn't impact the network at all!
- This seems wrong given that word order does have meaning in many languages, including English!



(slides/images C. Manning)

Transformer

Solution: Inject Order Information through Positional Encodings!



(slides/images C. Manning)

Transformer



Attention and Transformers

Representing the input order (positional encoding)

- Transformer is permutation invariant
- The transformer adds a vector to each input embedding.
- These vectors follow a specific pattern that the model learns
- Learned pattern helps model
 - to determine the position of each word, or
 - the distance between different words.

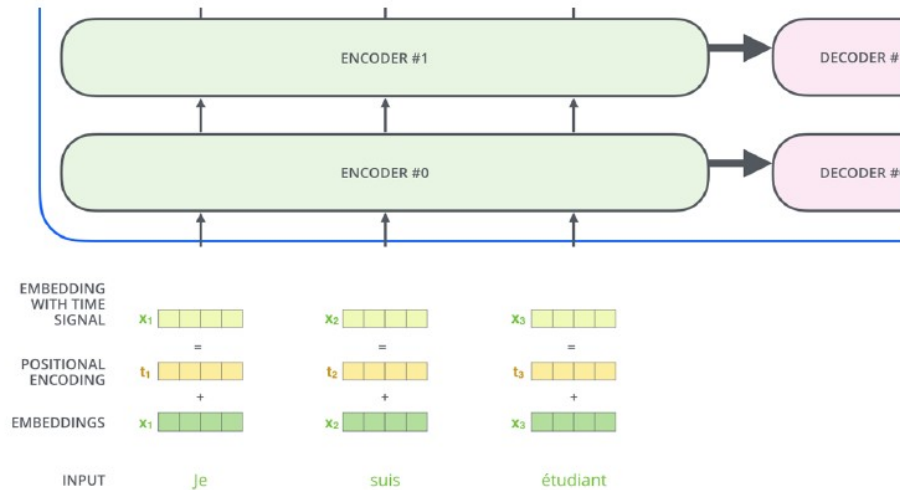


Transformer

02

Attention and Transformers

Representing the input order (positional encoding)



Transformer

Position Encoding

$$\vec{p}_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k + 1 \end{cases}$$

where

$$\omega_k = \frac{1}{10000^{2k/d}}$$

$$\vec{p}_t = \begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} \cdot t) \\ \cos(\omega_{d/2} \cdot t) \end{bmatrix}_{d \times 1}$$

Transformer

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i + 1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

Sequence

Index of token, k

Positional Encoding Matrix with $d=4, n=100$

		$i=0$	$i=0$	$i=1$	$i=1$
I	0	$P_{00}=\sin(0)$ = 0	$P_{01}=\cos(0)$ = 1	$P_{02}=\sin(0)$ = 0	$P_{03}=\cos(0)$ = 1
am	1	$P_{10}=\sin(1/1)$ = 0.84	$P_{11}=\cos(1/1)$ = 0.54	$P_{12}=\sin(1/10)$ = 0.10	$P_{13}=\cos(1/10)$ = 1.0
a	2	$P_{20}=\sin(2/1)$ = 0.91	$P_{21}=\cos(2/1)$ = -0.42	$P_{22}=\sin(2/10)$ = 0.20	$P_{23}=\cos(2/10)$ = 0.98
Robot	3	$P_{30}=\sin(3/1)$ = 0.14	$P_{31}=\cos(3/1)$ = -0.99	$P_{32}=\sin(3/10)$ = 0.30	$P_{33}=\cos(3/10)$ = 0.96

Positional Encoding Matrix for the sequence 'I am a robot'

Transformer

Fixing the first self-attention problem: **sequence order**

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

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

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

$$v_i = \tilde{v}_i + p_i$$

$$q_i = \tilde{q}_i + p_i$$

$$k_i = \tilde{k}_i + p_i$$

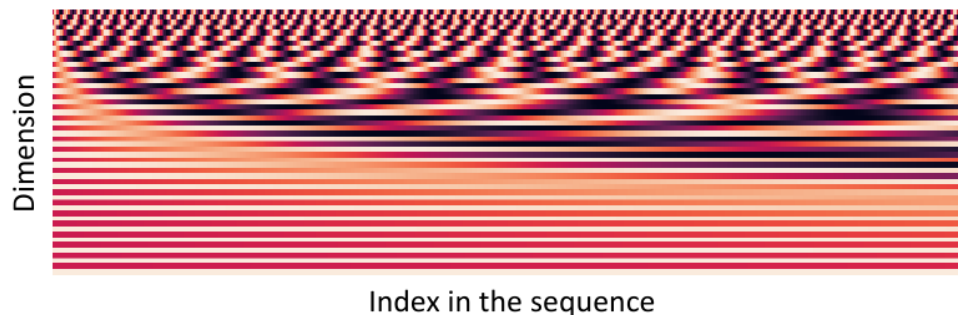
In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add...

Transformer

Position representation vectors through sinusoids (original)

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

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



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

Transformer

Extension: Self-Attention w/ Relative Position Encodings

Key Insight: The most salient position information is the relationship (e.g. “cat” is the word before “eat”) between words, rather than their absolute position (e.g. “cat” is word 2).

Original Self-Attention Output:

$$z_i = \sum_{j=1}^n \alpha_{ij} (x_j W^V)$$

where $\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^n \exp e_{ik}}$

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K)^T}{\sqrt{d_z}}$$

Relation-Aware Self-Attention Output:

$$z_i = \sum_{j=1}^n \alpha_{ij} (x_j W^V + a_{ij}^V)$$

where $\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^n \exp e_{ik}}$

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}$$

$$a_{ij}^K = w_{\text{clip}(j-i, k)}^K$$

$$a_{ij}^V = w_{\text{clip}(j-i, k)}^V$$

$$\text{clip}(x, k) = \max(-k, \min(k, x))$$

k	EN-DE BLEU
0	12.5
1	25.5
2	25.8
4	25.9
16	25.8
64	25.9
256	25.8

We then learn relative position representations
 $w^K = (w_{-k}^K, \dots, w_k^K)$ and $w^V = (w_{-k}^V, \dots, w_k^V)$

[Table and Equations From \[Shaw et al., 2018\]](#)

(slides/images C. Manning)

Transformer

Position Encoding

- Can also be learned
- Learn like other parameters

(slides/images C. Manning)

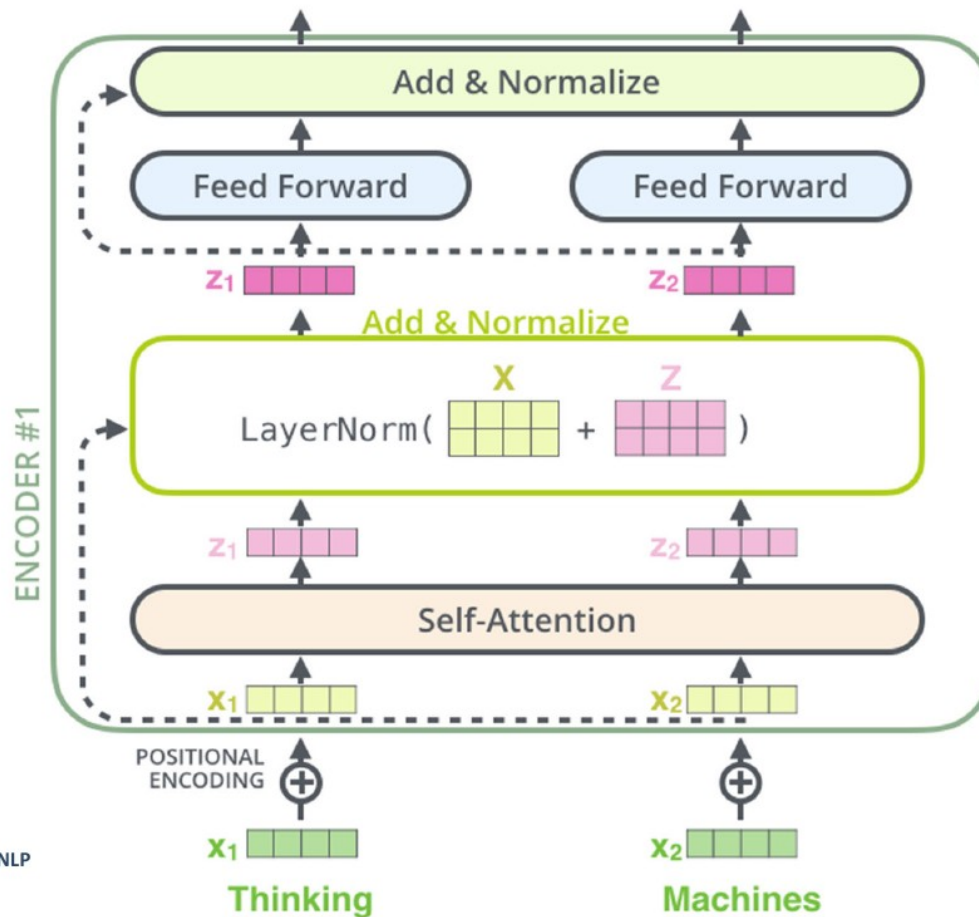


Transformer



Attention and Transformers

Add and Normalize



Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP



Transformer

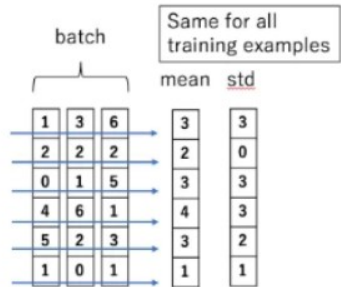


Attention and Transformers

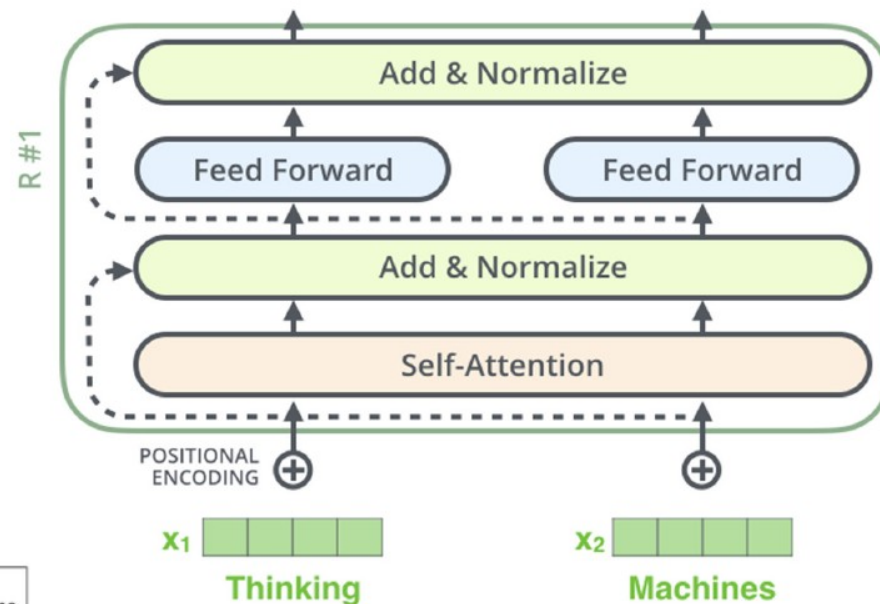
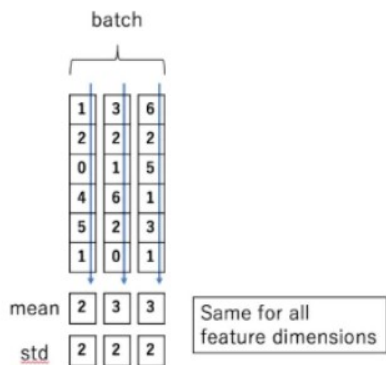
Layer Normalization (Hinton)

Layer normalization normalizes the inputs across the features.

Batch Normalization



Layer Normalization



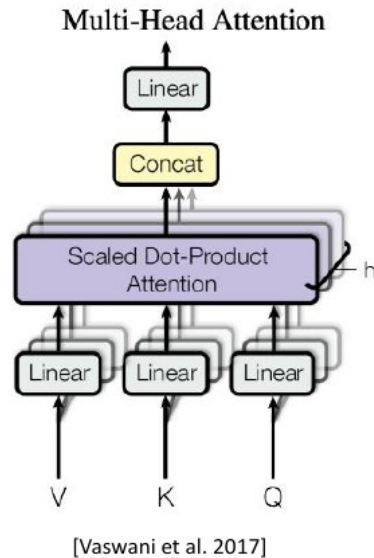
Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP



Transformer

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

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



Wizards of the Coast, Artist: Todd Lockwood

(slides/images C. Manning)

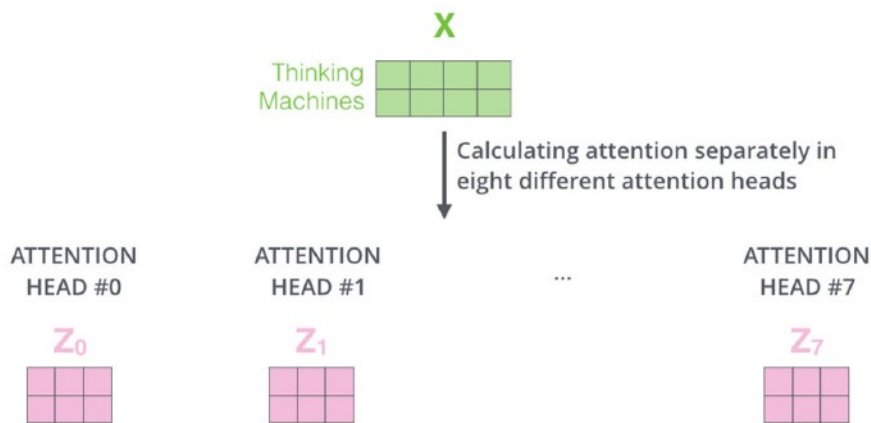
Transformer



Attention and Transformers

Multiple heads

1. It expands the model's ability to focus on different positions.
2. It gives the attention layer multiple "representation subspaces"



Transformer



Attention and Transformers

1) Concatenate all the attention heads

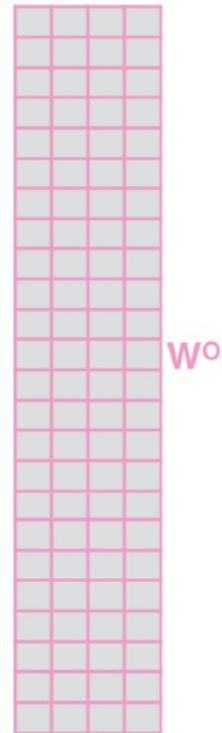


3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN



2) Multiply with a weight matrix W^O that was trained jointly with the model

X



Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP

Transformer

1) This is our input sentence*

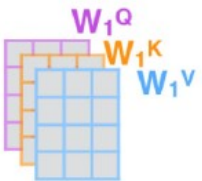
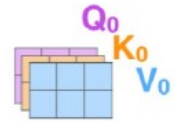
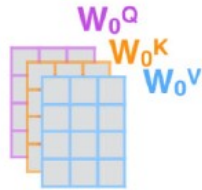
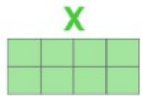
2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

4) Calculate attention using the resulting $Q/K/V$ matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

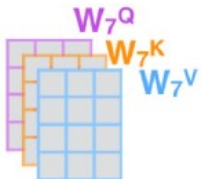
Thinking Machines



...

...

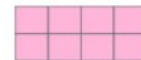
...



W^O

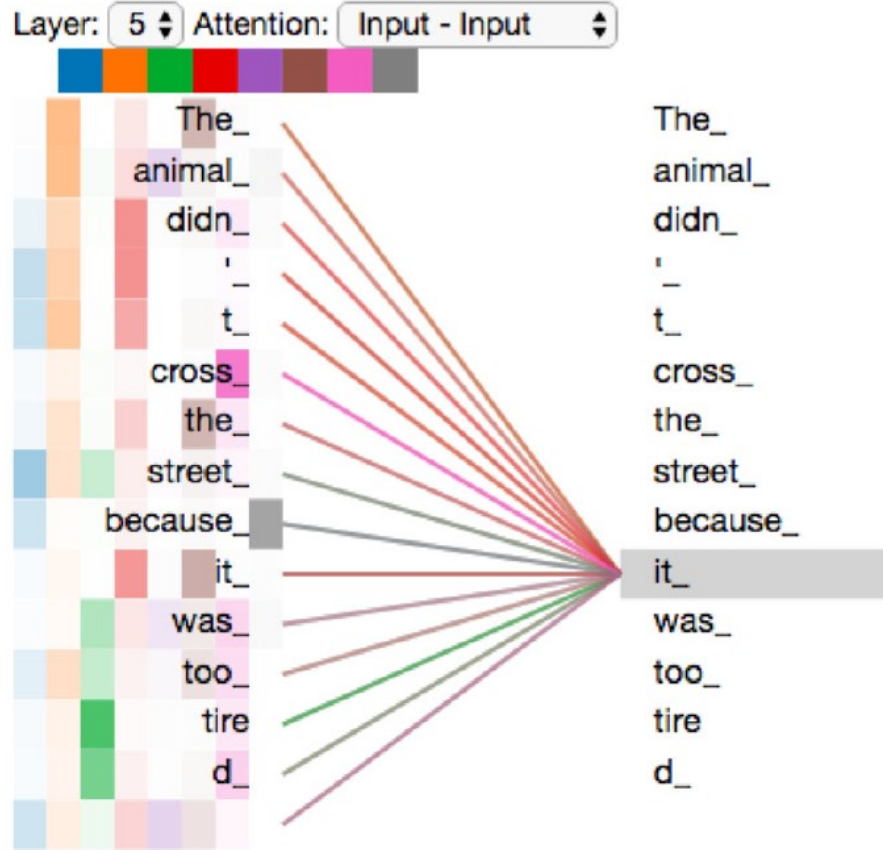


Z



Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP

Transformer

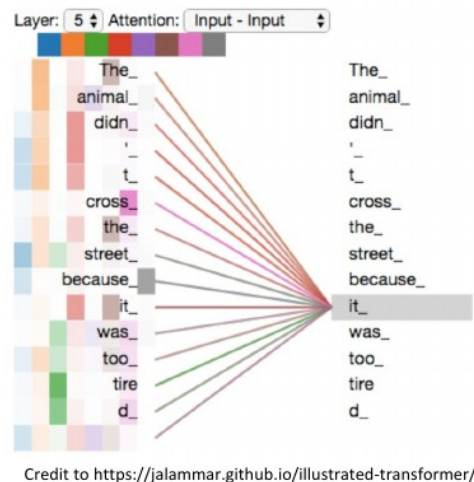


Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP

Transformer

The Transformer Encoder: Multi-headed Self-Attention

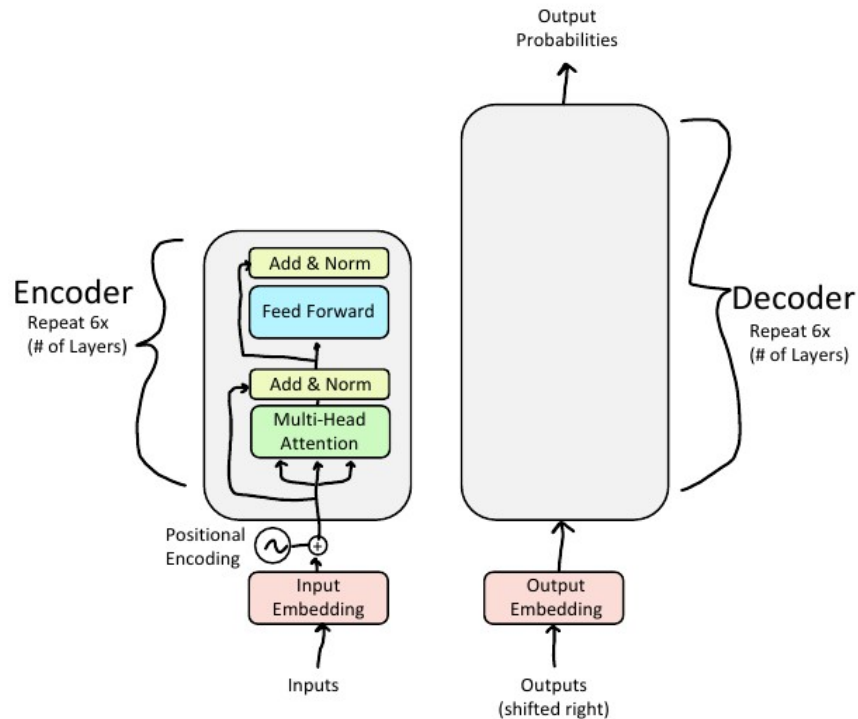
- What if we want to look in multiple places in the sentence at once?
 - For word i , self-attention “looks” where $x_i^T Q^T K x_j$ is high, but maybe we want to focus on different j for different reasons?
- We’ll define **multiple attention “heads”** through multiple Q,K,V matrices
- Let, $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and ℓ ranges from 1 to h .
- Each attention head performs attention independently:
 - $\text{output}_\ell = \text{softmax}(X Q_\ell K_\ell^T X^T) * X V_\ell$, where $\text{output}_\ell \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
 - $\text{output} = Y[\text{output}_1; \dots; \text{output}_h]$, where $Y \in \mathbb{R}^{d \times d}$
- Each head gets to “look” at different things, and construct value vectors differently.



(slides/images C. Manning)

Transformer

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

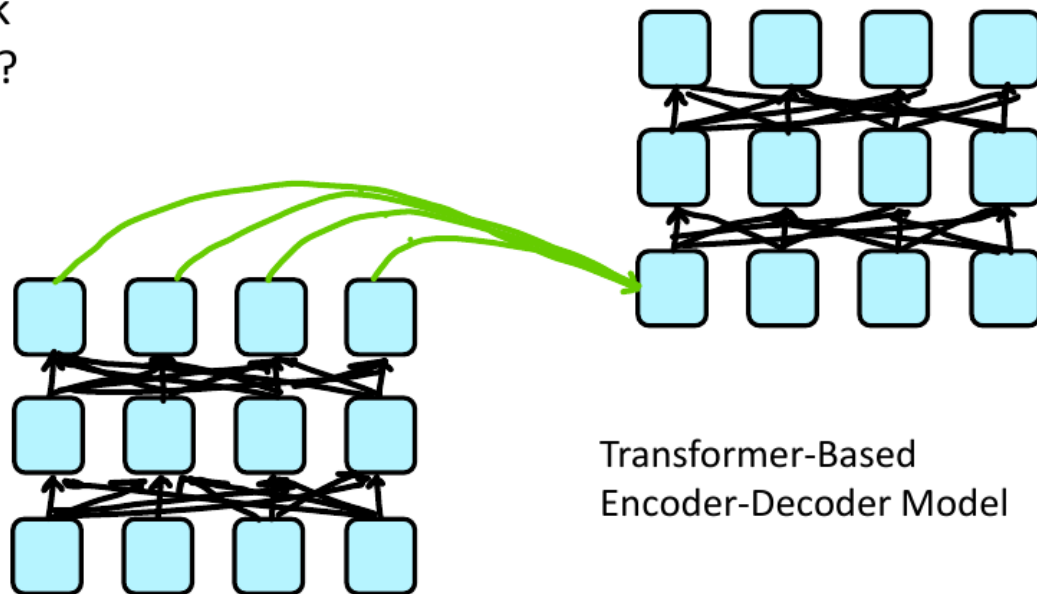


(slides/images C. Manning)

Transformer

Decoder: Masked Multi-Head Self-Attention

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



(slides/images C. Manning)

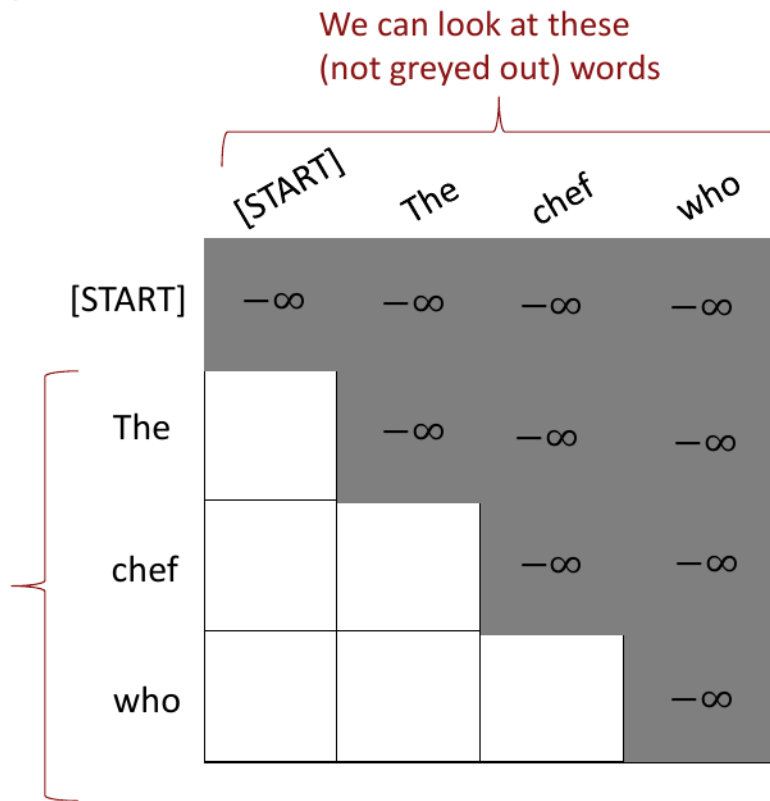
Transformer

Masking the future in self-attention

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

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

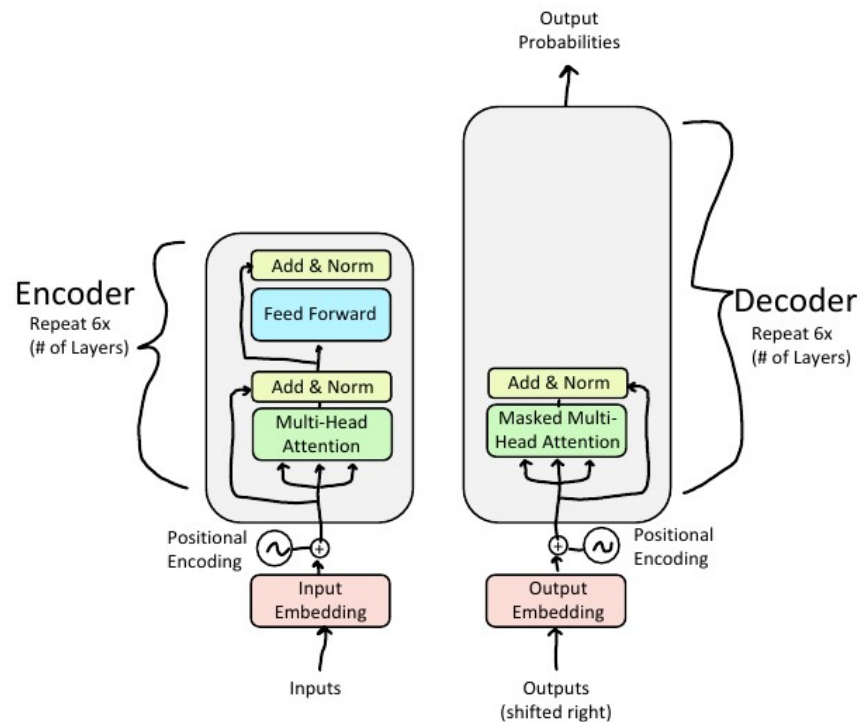
For encoding these words



(slides/images C. Manning)

Transformer

Decoder: Masked Multi-Headed Self-Attention

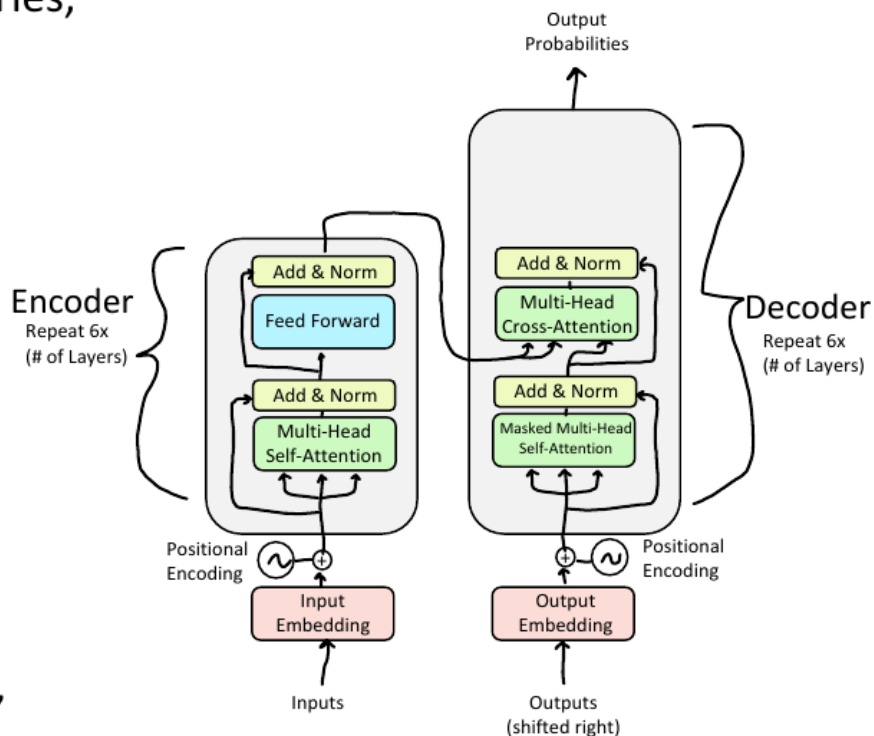


(slides/images C. Manning)

Transformer

Encoder-Decoder Attention

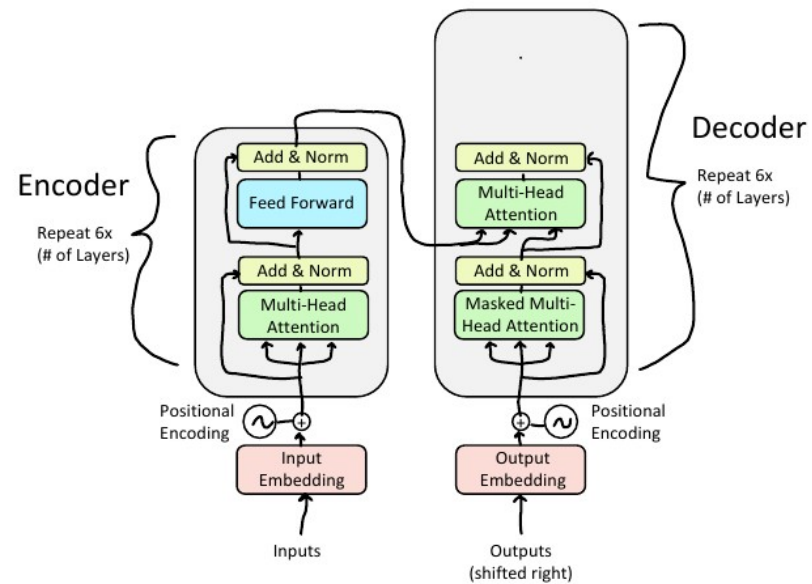
- We saw that self-attention is when keys, queries, and values come from the same source.
- In the decoder, we have attention that looks more like what we saw last week.
- Let h_1, \dots, h_T be **output** vectors from the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let z_1, \dots, z_T be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the **encoder** (like a memory):
 - $k_i = Kh_i, v_i = Vh_i$.
- And the queries are drawn from the **decoder**, $q_i = Qz_i$.



(slides/images C. Manning)

Transformer

Decoder: Finishing touches!

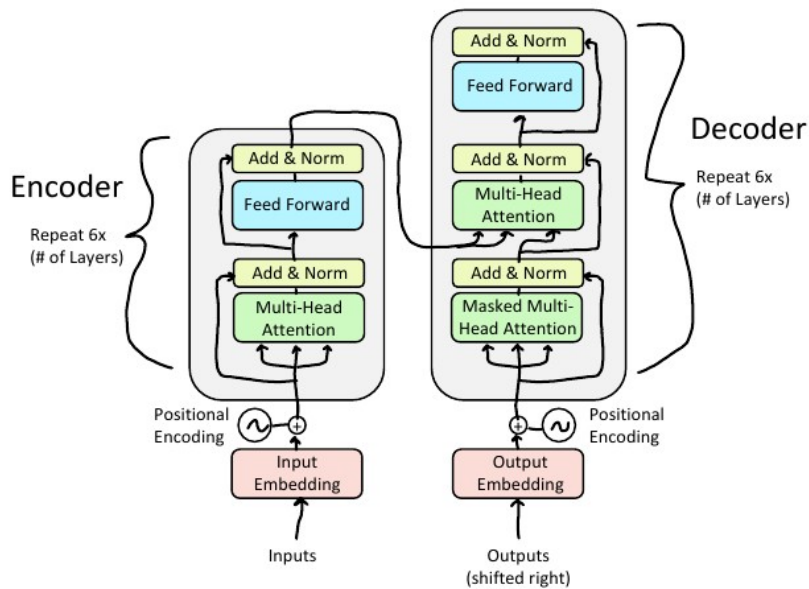


(slides/images C. Manning)

Transformer

Decoder: Finishing touches!

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

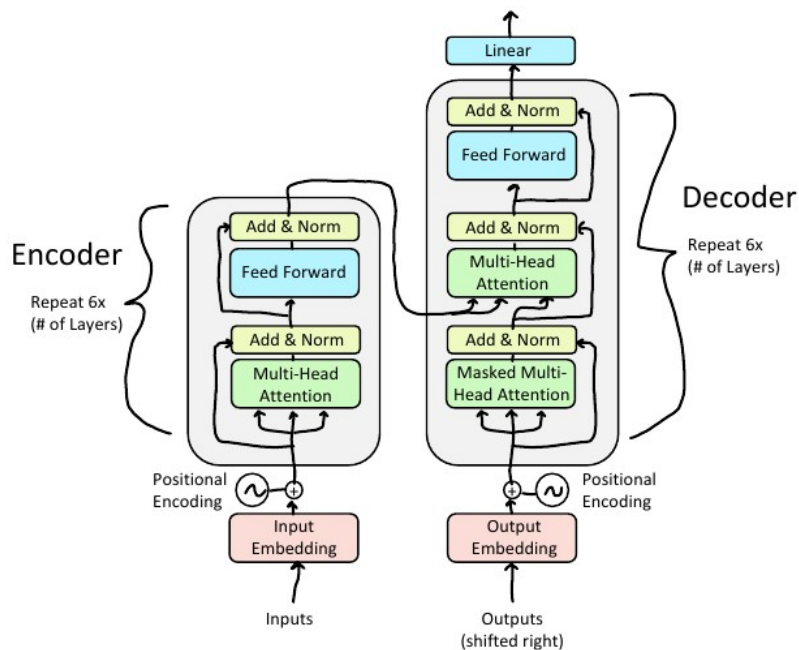


(slides/images C. Manning)

Transformer

Decoder: Finishing touches!

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

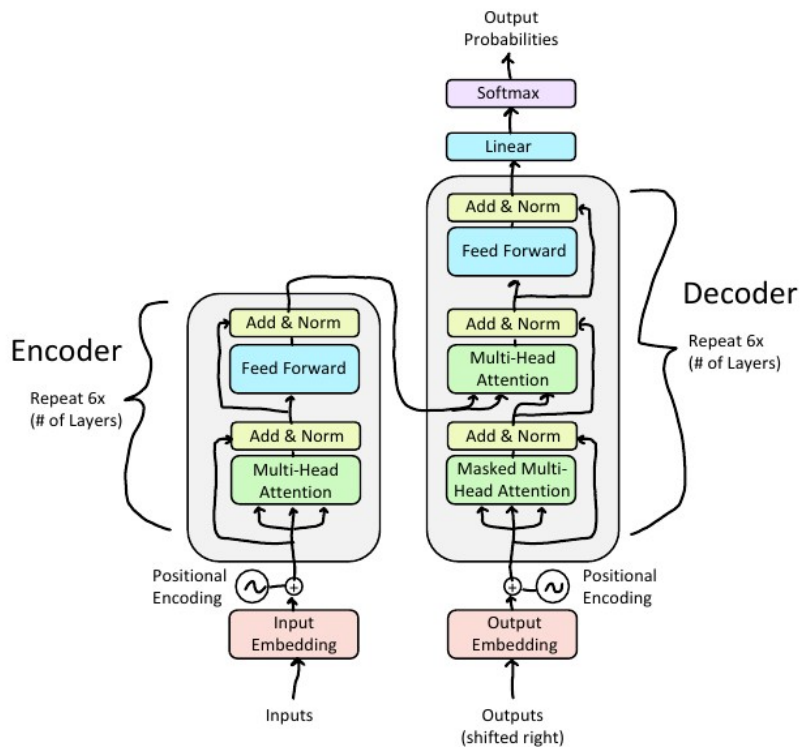


(slides/images C. Manning)

Transformer

Decoder: Finishing touches!

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



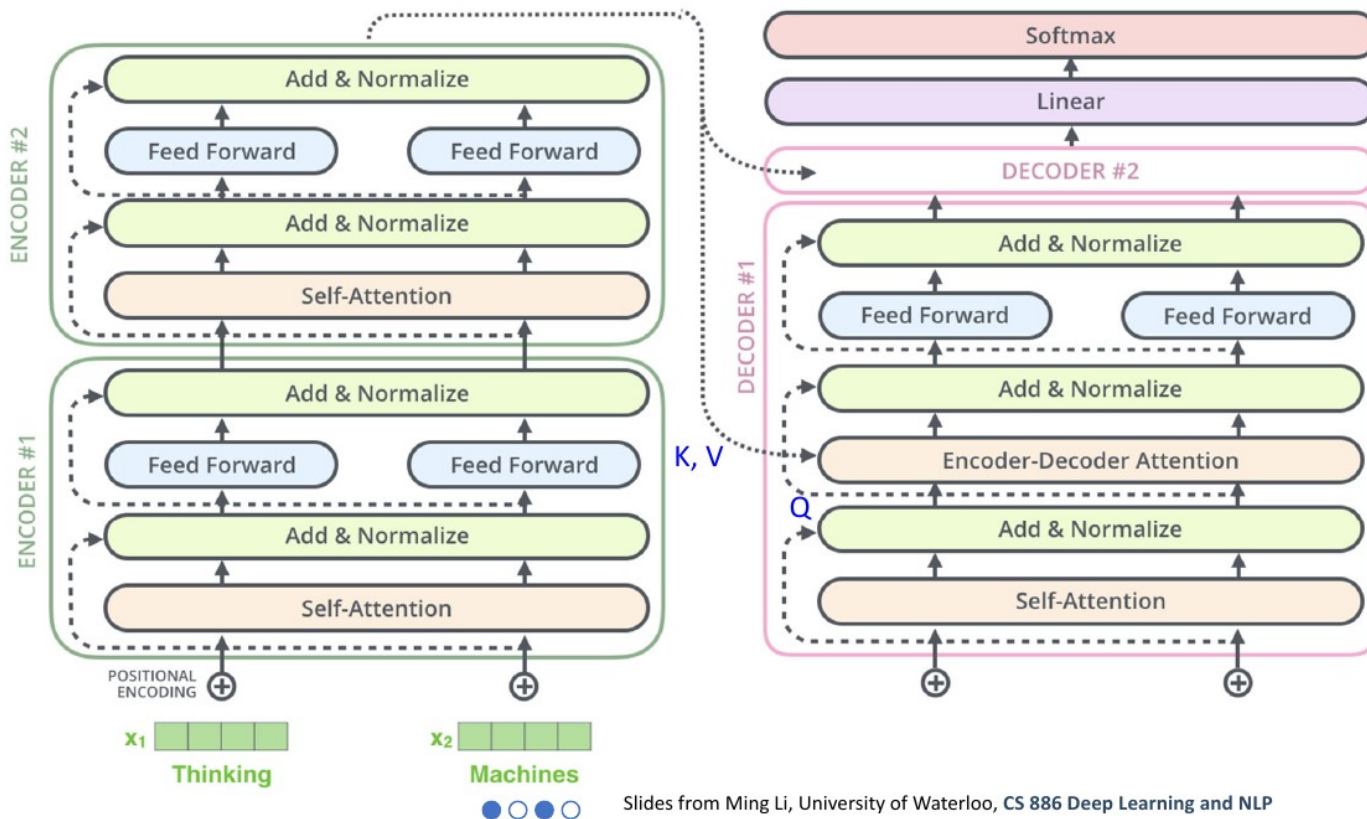
(slides/images C. Manning)

Transformer

02

Attention and Transformers

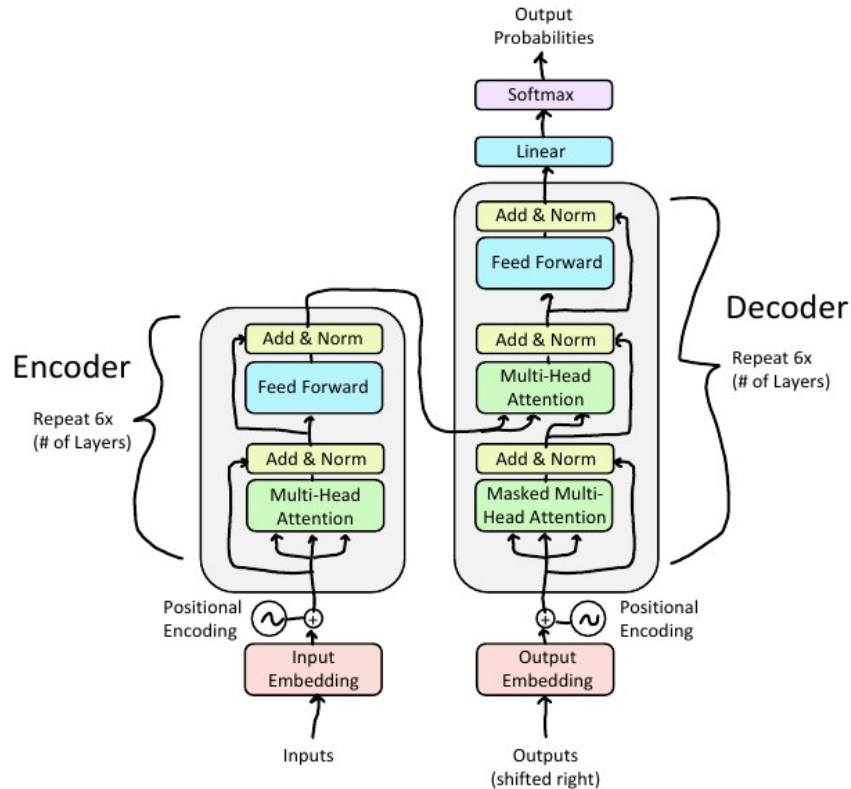
The complete transformer



Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP

Transformer

Recap of Transformer Architecture



(slides/images C. Manning)

Transformer

Transformer

- Used for modeling long dependencies between input sequence elements
- Supports parallel processing of sequence as compared to RNN (e.g. LSTM)
- Allows processing multiple modalities
 - (e.g., images, videos, text and speech) using similar processing blocks
- Typically, pre-trained using pretext tasks on largescale (unlabeled) datasets
- Demonstrates excellent scalability to very large networks and huge datasets.

- **GPT-4** (Generative Pretrained Transformer)

(slides/images M. Shah)



Transformer

Vision Applications

- Recognition tasks (e.g., image classification, object detection, action recognition, and segmentation),
- Generative modeling, multi-modal tasks (e.g., visual-question answering, visual reasoning, and visual grounding),
- Video processing (e.g., activity recognition, video forecasting),
- Low-level vision (e.g., image super-resolution, image enhancement, and colorization)
- 3D analysis (e.g., point cloud classification and segmentation)

(slides/images M. Shah)



Transformer

Natural Language Processing

- BERT (Bidirectional Encoder Representations from Transformers),
- GPT (Generative Pre-trained **Transformer**) v1-4,
- RoBERTa (Robustly Optimized BERT Pre-training)
- T5 (Text-to-Text Transfer Transformer)

(slides/images M. Shah)



Enquanto isso...



© marketoonist.com

<https://marketoonist.com/2023/06/impact-of-chatgpt.html>

Referências desta aula

- Capítulos 7 e 9 “Deep Learning for NLP and Speech Recognition”, Uday Kamath, John Liu, and James Whitaker