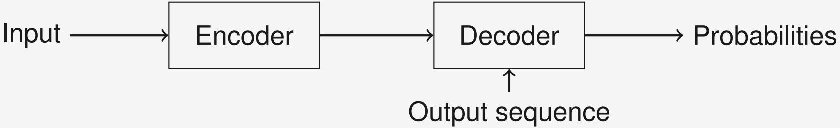
Self-attention: an introduction

A series of videos on transformers

Lennart Svensson

TRANSFORMER

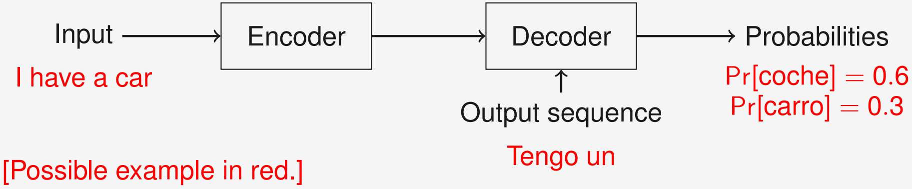
High-level transformer architecture



- The encoder-decoder structure is standard in machine translations.

TRANSFORMER

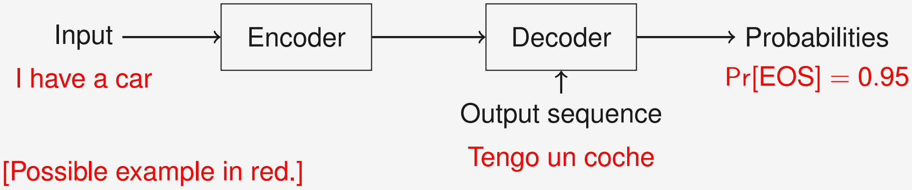
High-level transformer architecture



- The encoder-decoder structure is standard in machine translations.

TRANSFORMER

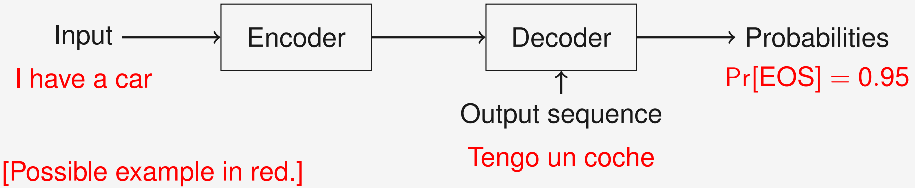
High-level transformer architecture



- The encoder-decoder structure is standard in machine translations.

TRANSFORMER

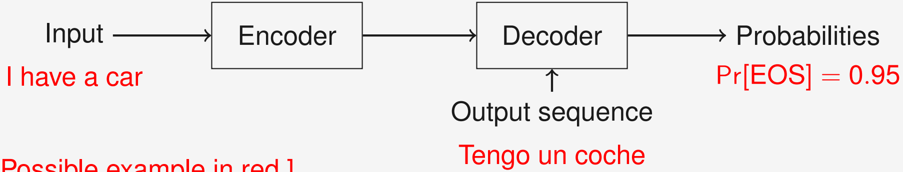
High-level transformer architecture



- The encoder-decoder structure is standard in machine translations.
- What’s new? Encoder and decoder use **deep self-attention networks**.

TRANSFORMER

High-level transformer architecture



[Possible example in red.]

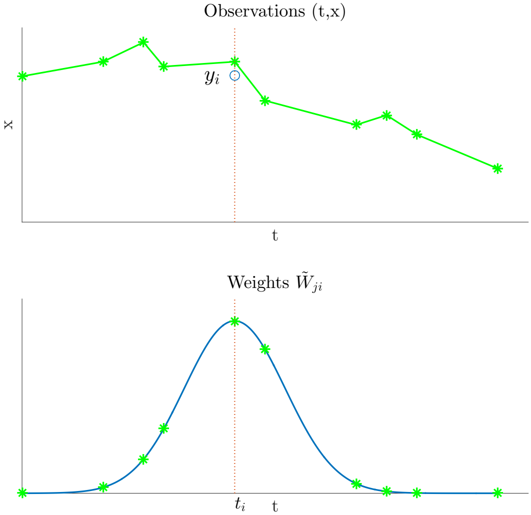
- The encoder-decoder structure is standard in machine translations.
- What's new? Encoder and decoder use **deep self-attention networks**.
- Encoder and decoder are also used in other contexts.

WEIGHTED AVERAGES

- Compute weighted average

$$y_i = \sum_j W_{ji} x_j.$$

- Weights W_{ji}?



WEIGHTED AVERAGES

- Compute weighted average

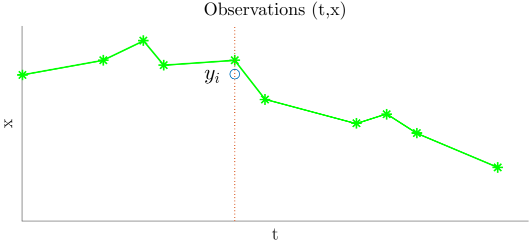
$$y_i = \sum_j W_{ji} x_j.$$

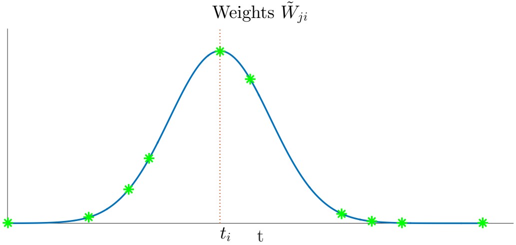
- Weights W_{ji}?
- Decrease with $|t_j - t_i|$:

$$\tilde{W}_{ji} = \mathcal{N}(t_j; t_i, 1).$$

- Normalize weights to sum to 1:

$$W_{ji} = \frac{\tilde{W}_{ji}}{\sum_r \tilde{W}_{ri}}.$$





WEIGHTED AVERAGES OF WORDS

Emma hates games but she is a great friend

WEIGHTED AVERAGES OF WORDS

	Emma	hates	games	but	she	is	a	great	friend
Word vectors:	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

WEIGHTED AVERAGES OF WORDS

	Emma	hates	games	but	she	is	a	great	friend
Word vectors:	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

- New vector representing "friend":

$$y_9 = \sum_i w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_9 x_9.$$

WEIGHTED AVERAGES OF WORDS

	Emma	hates	games	but	she	is	a	great	friend
Word vectors:	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

- New vector representing "friend":

$$y_9 = \sum_i w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_9 x_9.$$

WEIGHTED AVERAGES OF WORDS

	Emma	hates	games	but	she	is	a	great	friend
Word vectors:	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

- New vector representing “friend”:

$$y_9 = \sum_i w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_9 x_9.$$

- Weights?

WEIGHTED AVERAGES OF WORDS

Who does “friend” refer to?

	Emma	hates	games	but	she	is	a	great	friend
Word vectors:	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

- New vector representing “friend”:

$$y_9 = \sum_i w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_9 x_9.$$

- Weights?

WEIGHTED AVERAGES OF WORDS

Who does "friend" refer to?

	Emma	hates	games	but	she	is	a	great	friend
Word vectors:	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

- New vector representing "friend":

$$y_9 = \sum_i w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_9 x_9.$$

- Weights? **A first idea**:

$$z_1 = x_1^\top x_9,\ z_2 = x_2^\top x_9, \ldots \qquad \text{Similar words } \Rightarrow \text{large weight!}$$

$$[w_1, w_2, \ldots, w_9] = \text{softmax}(z_1, z_2, \ldots, z_9)$$

WEIGHTS IN SELF-ATTENTION

- To compute weights we compute new vectors

$$\text{Query:} \qquad q_9 = W_Q x_9$$

$$\text{Keys:} \qquad k_1 = W_K x_1,\ k_2 = W_K x_2,\ \ldots,\ k_9 = W_K x_9$$

WEIGHTS IN SELF-ATTENTION

- To compute weights we compute new vectors

$$\text{Query:} \qquad q_9 = W_Q x_9$$

$$\text{Keys:} \qquad k_1 = W_K x_1, \; k_2 = W_K x_2, \; \ldots, \; k_9 = W_K x_9$$

- Weights are obtained from products

$$z_1 = k_1^\mathsf{T} q_9, \;\; z_2 = k_2^\mathsf{T} q_9, \; \ldots, \; z_9 = k_9^\mathsf{T} q_9$$

$$[w_1, w_2, \ldots, w_9] = \mathrm{softmax}(z_1, z_2, \ldots, z_9)$$

WEIGHTS IN SELF-ATTENTION

- To compute weights we compute new vectors

$$\text{Query:} \qquad q_9 = W_Q x_9$$

$$\text{Keys:} \qquad k_1 = W_K x_1, \; k_2 = W_K x_2, \; \dots, \; k_9 = W_K x_9$$

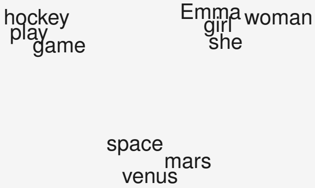
- Weights are obtained from products

$$z_1 = k_1^T q_9, \;\; z_2 = k_2^T q_9, \; \dots, \; z_9 = k_9^T q_9$$

$$[w_1, w_2, \dots, w_9] = \text{softmax}(z_1, z_2, \dots, z_9)$$

Large weights for “Emma” and “she”?

- **Goal**: large z_1 and z_5.



WEIGHTS IN SELF-ATTENTION

- To compute weights we compute new vectors

$$\text{Query:} \qquad q_9 = W_Q x_9$$

$$\text{Keys:} \qquad k_1 = W_K x_1,\ k_2 = W_K x_2,\ \ldots,\ k_9 = W_K x_9$$

- Weights are obtained from products

$$z_1 = k_1^T q_9,\ \ z_2 = k_2^T q_9,\ \ldots,\ z_9 = k_9^T q_9$$

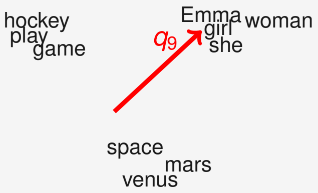
$$[w_1, w_2, \ldots, w_9] = \text{softmax}(z_1, z_2, \ldots, z_9)$$

Large weights for “Emma” and “she”?

- **Goal**: large z_1 and z_5.
- Set $W_K = I$ and W_Q: q_9 in figure.
- This gives large dot products

$$z_1 = k_1^T q_9 = x_1^T q_9$$

$$z_5 = k_5^T q_9 = x_5^T q_9$$



SELF-ATTENTION WEIGHTS IN OUR EXAMPLE

Who does "friend" refer to?

Emma	hates	games	but	she	is	a	great	friend
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

Word vectors: $x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8 \quad x_9$

- Weights in self-attention:

Query: $q_9 = W_Q x_9,$

Keys: $k_1 = W_K x_1,\ k_2 = W_K x_2,\ \ldots,\ k_9 = W_K x_9$

Weights: $$\begin{cases} z_1 = k_1^{\mathsf{T}} q_9,\ \ z_2 = k_2^{\mathsf{T}} q_9,\ \ldots,\ z_9 = k_9^{\mathsf{T}} q_9 \\ [w_1, w_2, \ldots, w_9] = \mathrm{softmax}(z_1, z_2, \ldots, z_9) \end{cases}$$

- New vector representing "friend":

$$y_9 = \sum_i w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_9 x_9.$$

SELF-ATTENTION WEIGHTS IN OUR EXAMPLE

	Emma	hates	games	but	she	is	a	great	friend
Word vectors:	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

- Weights in self-attention:

$$\begin{aligned} \text{Query:} \quad & q_3 = W_Q x_3, \\ \text{Keys:} \quad & k_1 = W_K x_1,\ k_2 = W_K x_2,\ \ldots,\ k_9 = W_K x_9 \\ \text{Weights:} \quad & \begin{cases} z_1 = k_1^\top q_3,\ \ z_2 = k_2^\top q_3,\ \ldots,\ z_9 = k_9^\top q_3 \\ [w_1, w_2, \ldots, w_9] = \text{softmax}(z_1, z_2, \ldots, z_9) \end{cases} \end{aligned}$$

- New vector representing “games”:

$$y_3 = \sum_i w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_9 x_9.$$

Self-attention: complete description

A series of videos on transformers

Lennart Svensson

SELF-ATTENTION: VECTOR DESCRIPTION

- Let n : $\sharp$ words in sentence, d : length of x_i.

SELF-ATTENTION: VECTOR DESCRIPTION

- Let n : $\sharp$ words in sentence, d : length of x_i.

Self-attention: vector notation

- For all $i \in \{1, 2, \ldots, n\}$:

$$\begin{aligned} \text{Query:} \quad & q_i = W_Q x_i, \\ \text{Keys:} \quad & k_i = W_K x_i, \\ \text{Values:} \quad & v_i = W_V x_i. \end{aligned}$$

SELF-ATTENTION: VECTOR DESCRIPTION

- Let n : $\sharp$ words in sentence, d : length of x_i.

Self-attention: vector notation

- For all $i \in \{1, 2, \ldots, n\}$:

$$\begin{aligned} \text{Query:} \qquad & q_i = W_Q x_i, \\ \text{Keys:} \qquad & k_i = W_K x_i, \\ \text{Values:} \qquad & v_i = W_V x_i. \end{aligned}$$

- For all $i, j \in \{1, 2, \ldots, n\}$:

$$Z_{ji} = k_j^{\mathsf{T}} q_i / \sqrt{d}.$$

SELF-ATTENTION: VECTOR DESCRIPTION

- Let n : $\sharp$ words in sentence, d : length of x_i.

Self-attention: vector notation

- For all $i \in \{1, 2, \ldots, n\}$:

$$\begin{aligned} \text{Query:} \quad & q_i = W_Q x_i, \\ \text{Keys:} \quad & k_i = W_K x_i, \\ \text{Values:} \quad & v_i = W_V x_i. \end{aligned}$$

- For all $i, j \in \{1, 2, \ldots, n\}$:

$$Z_{ji} = k_j^{\mathsf{T}} q_i / \sqrt{d}.$$

- For all $i \in \{1, 2, \ldots, n\}$:

$$\begin{aligned} \text{Weights:} \quad & [W_{1i}, W_{2i}, \ldots, W_{ni}] = \operatorname{softmax}(Z_{1i}, Z_{2i}, \ldots, Z_{ni}) \\ \text{New embedding:} \quad & y_i = \sum_{j=1}^{n} v_j W_{ji}. \end{aligned}$$

SELF-ATTENTION MAPS SETS TO SETS

- Order does not matter. It maps sets to sets!

SELF-ATTENTION MAPS SETS TO SETS

- Order does not matter. It maps sets to sets!

Toy example

Does the word order influence the output embeddings?

$$
\begin{array}{rll}
\text{he} & \rightarrow x_1 & \rightarrow q_1 = W_Q x_1,\ k_1 = W_K x_1,\ v_1 = W_v x_1 \\
\text{is} & \rightarrow x_2 & \rightarrow q_2 = W_Q x_2,\ k_2 = W_K x_2,\ v_2 = W_v x_2 \\
\text{old} & \rightarrow x_3 & \rightarrow q_3 = W_Q x_3,\ k_3 = W_K x_3,\ v_3 = W_v x_3
\end{array}
$$

SELF-ATTENTION MAPS SETS TO SETS

- Order does not matter. It maps sets to sets!

Toy example

Does the word order influence the output embeddings?

$$
\begin{array}{rll}
\text{he} & \rightarrow x_{\text{he}} & \rightarrow q_{\text{he}} = W_Q x_{\text{he}}, \quad k_{\text{he}} = W_K x_{\text{he}}, \quad v_{\text{he}} = W_v x_{\text{he}} \\
\text{is} & \rightarrow x_{\text{is}} & \rightarrow q_{\text{is}} = W_Q x_{\text{is}}, \quad k_{\text{is}} = W_K x_{\text{is}}, \quad v_{\text{is}} = W_v x_{\text{is}} \\
\text{old} & \rightarrow x_{\text{old}} & \rightarrow q_{\text{old}} = W_Q x_{\text{old}}, \quad k_{\text{old}} = W_K x_{\text{old}}, \quad v_{\text{old}} = W_v x_{\text{old}}
\end{array}
$$

SELF-ATTENTION MAPS SETS TO SETS

- Order does not matter. It maps sets to sets!

Toy example

Does the word order influence the output embeddings?

$$
\begin{array}{lll}
\text{he} & \rightarrow x_{\text{he}} & \rightarrow q_{\text{he}} = W_Q x_{\text{he}}, \quad k_{\text{he}} = W_K x_{\text{he}}, \quad v_{\text{he}} = W_v x_{\text{he}} \\
\text{is} & \rightarrow x_{\text{is}} & \rightarrow q_{\text{is}} = W_Q x_{\text{is}}, \quad k_{\text{is}} = W_K x_{\text{is}}, \quad v_{\text{is}} = W_v x_{\text{is}} \\
\text{old} & \rightarrow x_{\text{old}} & \rightarrow q_{\text{old}} = W_Q x_{\text{old}}, \quad k_{\text{old}} = W_K x_{\text{old}}, \quad v_{\text{old}} = W_v x_{\text{old}}
\end{array}
$$

Weights for "he": $Z_{\text{he,he}} = k_{\text{he}}^{\mathsf{T}} q_{\text{he}}, \quad Z_{\text{is,he}} = k_{\text{is}}^{\mathsf{T}} q_{\text{he}}, \quad Z_{\text{old,he}} = k_{\text{old}}^{\mathsf{T}} q_{\text{he}}$

$$
\begin{bmatrix} W_{\text{he,he}} \\ W_{\text{is,he}} \\ W_{\text{old,he}} \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} Z_{\text{he,he}} \\ Z_{\text{is,he}} \\ Z_{\text{old,he}} \end{bmatrix} \right)
$$

SELF-ATTENTION MAPS SETS TO SETS

- Order does not matter. It maps sets to sets!

Toy example

Does the word order influence the output embeddings?

$$
\begin{array}{lll}
\text{he} & \rightarrow x_{\text{he}} & \rightarrow q_{\text{he}} = W_Q x_{\text{he}}, \quad k_{\text{he}} = W_K x_{\text{he}}, \quad v_{\text{he}} = W_v x_{\text{he}} \\
\text{is} & \rightarrow x_{\text{is}} & \rightarrow q_{\text{is}} = W_Q x_{\text{is}}, \quad k_{\text{is}} = W_K x_{\text{is}}, \quad v_{\text{is}} = W_v x_{\text{is}} \\
\text{old} & \rightarrow x_{\text{old}} & \rightarrow q_{\text{old}} = W_Q x_{\text{old}}, \quad k_{\text{old}} = W_K x_{\text{old}}, \quad v_{\text{old}} = W_v x_{\text{old}}
\end{array}
$$

Weights for "he": $Z_{\text{he,he}} = k_{\text{he}}^{\mathsf{T}} q_{\text{he}}, \quad Z_{\text{is,he}} = k_{\text{is}}^{\mathsf{T}} q_{\text{he}}, \quad Z_{\text{old,he}} = k_{\text{old}}^{\mathsf{T}} q_{\text{he}}$

$$
\begin{bmatrix} W_{\text{he,he}} \\ W_{\text{is,he}} \\ W_{\text{old,he}} \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} Z_{\text{he,he}} \\ Z_{\text{is,he}} \\ Z_{\text{old,he}} \end{bmatrix} \right)
$$

$$
y_{\text{he}} = v_{\text{he}} W_{\text{he,he}} + v_{\text{is}} W_{\text{is,he}} + v_{\text{old}} W_{\text{old,he}}
$$

SELF-ATTENTION: MATRIX DESCRIPTION

- Introducing matrix notations:

$$X = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}, \qquad Y = \begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix},$$

$$Q = \begin{bmatrix} q_1 & q_2 & \dots & q_n \end{bmatrix}, \qquad K = \begin{bmatrix} k_1 & k_2 & \dots & k_n \end{bmatrix}, \qquad V = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$$

SELF-ATTENTION: MATRIX DESCRIPTION

- Introducing matrix notations:

$$X = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}, \qquad Y = \begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix},$$

$$Q = \begin{bmatrix} q_1 & q_2 & \dots & q_n \end{bmatrix}, \qquad K = \begin{bmatrix} k_1 & k_2 & \dots & k_n \end{bmatrix}, \qquad V = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$$

Self-attention: matrix notation

- Calculations without for-loops:

$$\begin{aligned} \text{Query:} \qquad & Q = W_Q X, \\ \text{Keys:} \qquad & K = W_K X, \\ \text{Values:} \qquad & V = W_V X \\ & Z = K^{\mathsf{T}} Q / \sqrt{d}. \end{aligned}$$

- Weights and averages using columnwise softmax:

$$\begin{aligned} \text{Weights:} \qquad & W = \operatorname{softmax}(Z) \\ \text{New embedding:} \qquad & Y = VW. \end{aligned}$$

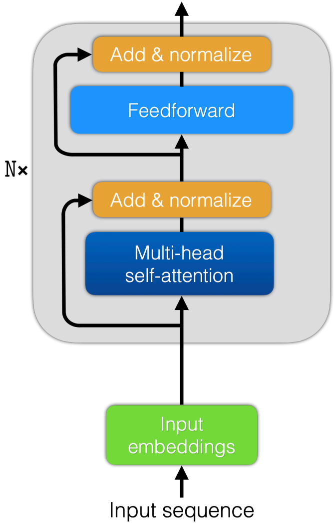
Encoder

A series of videos on transformers

Lennart Svensson

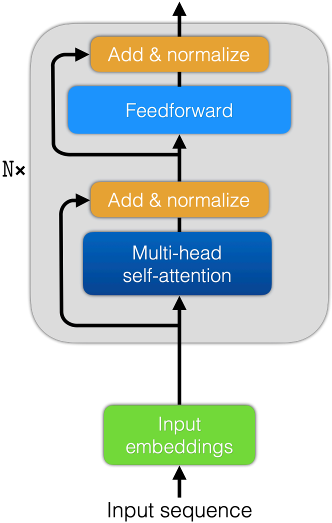
ENCODER OVERVIEW

- The encoder stacks *N* encoder blocks.



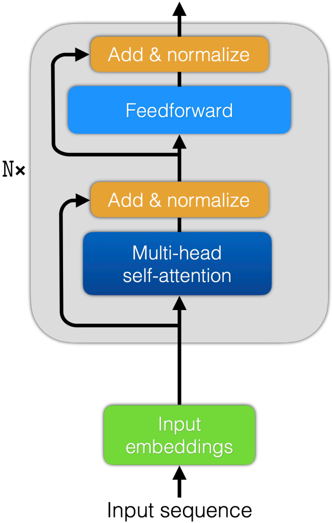
ENCODER OVERVIEW

- The encoder stacks *N* encoder blocks.
- Each encoder block maintains shape (#vectors, vector length).



ENCODER OVERVIEW

- The encoder stacks *N* encoder blocks.
- Each encoder block maintains shape (#vectors, vector length).
- This video briefly explains all components:
    - input embeddings with positional encodings,
    - multi-head self-attention,
    - add & normalize, and
    - feedforward networks.



INPUT EMBEDDINGS AND POSITIONAL ENCODINGS (1)

- Encoder blocks map sets to sets.

INPUT EMBEDDINGS AND POSITIONAL ENCODINGS (1)

- Encoder blocks map sets to sets.
- However, word order is not irrelevant.

    They are rich, but are they happy?
    They are happy, but are they rich?

INPUT EMBEDDINGS AND POSITIONAL ENCODINGS (1)

- Encoder blocks map sets to sets.
- However, word order is not irrelevant.

  They are rich, but are they happy?
  They are happy, but are they rich?

- Solution: encode position in input vectors!

INPUT EMBEDDINGS AND POSITIONAL ENCODINGS (2)

- A small example:

Input: He is happy!

INPUT EMBEDDINGS AND POSITIONAL ENCODINGS (2)

- A small example:

Input:	He is happy!			
Words in vocabulary:	He_	is_	happy	!

INPUT EMBEDDINGS AND POSITIONAL ENCODINGS (2)

- A small example:

Input:	He is happy!			
Words in vocabulary:	He_	is_	happy	!
One-hot encodings:	t_1	t_2	t_3	t_4

INPUT EMBEDDINGS AND POSITIONAL ENCODINGS (2)

- A small example:

Input:	He is happy!			
Words in vocabulary:	He_	is_	happy	!
One-hot encodings:	t_1	t_2	t_3	t_4

Positional one-hot: $p_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \end{bmatrix}$ $p_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \end{bmatrix}$ $p_3 = \begin{bmatrix} 0 \\ 0 \\ \vdots \end{bmatrix}$ $p_4 = \begin{bmatrix} 0 \\ 0 \\ \vdots \end{bmatrix}$

INPUT EMBEDDINGS AND POSITIONAL ENCODINGS (2)

- A small example:

Input:	He is happy!			
Words in vocabulary:	He_	is_	happy	!
One-hot encodings:	t_1	t_2	t_3	t_4

Positional one-hot: $p_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \end{bmatrix}$ $p_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \end{bmatrix}$ $p_3 = \begin{bmatrix} 0 \\ 0 \\ \vdots \end{bmatrix}$ $p_4 = \begin{bmatrix} 0 \\ 0 \\ \vdots \end{bmatrix}$

- Input embeddings (vectors), $i = 1, 2, 3, 4$:

$$x_i = \underbrace{E\, t_i}_{\text{Word embedding}} + \underbrace{P\, p_i}_{\text{Positional encoding}}$$

INPUT EMBEDDINGS AND POSITIONAL ENCODINGS (2)

- A small example:

Input:	He is happy!			
Words in vocabulary:	He_	is_	happy	!
One-hot encodings:	t_1	t_2	t_3	t_4

Positional one-hot: $p_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \end{bmatrix}$ $p_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \end{bmatrix}$ $p_3 = \begin{bmatrix} 0 \\ 0 \\ \vdots \end{bmatrix}$ $p_4 = \begin{bmatrix} 0 \\ 0 \\ \vdots \end{bmatrix}$

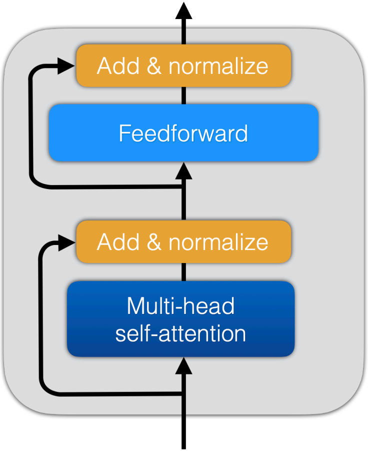
- Input embeddings (vectors), $i = 1, 2, 3, 4$:

$$x_i = \underbrace{E\, t_i}_{\text{Word embedding}} + \underbrace{P\, p_i}_{\text{Positional encoding}}$$

- Learnable parameters

E: $d\times$ size of vocabulary, P: $d\times$ max length of sequence.

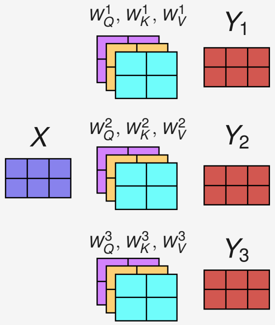
MULTI-HEAD SELF-ATTENTION



MULTI-HEAD SELF-ATTENTION

- h parallel self-attention blocks/heads.
- Blocks have identical structure but different parameters.

Example: $n = 3$, $d = 2$, $h = 3$

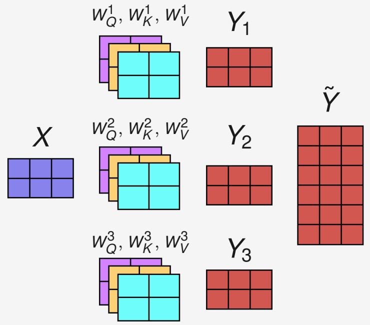


MULTI-HEAD SELF-ATTENTION

- h parallel self-attention blocks/heads.
- Blocks have identical structure but different parameters.
- Output vectors from different heads are concatenated (for each word)

$$\tilde{Y} = \begin{bmatrix} Y_1 \\ \vdots \\ Y_h \end{bmatrix}.$$

Example: $n = 3$, $d = 2$, $h = 3$



MULTI-HEAD SELF-ATTENTION

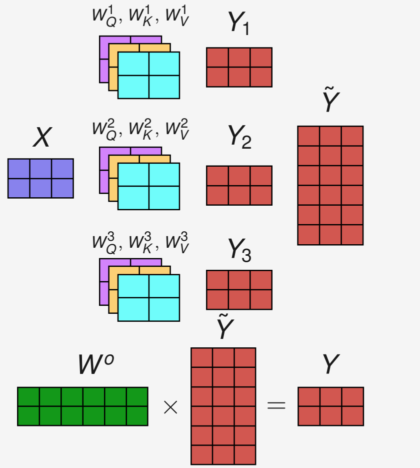
- h parallel self-attention blocks/heads.
- Blocks have identical structure but different parameters.
- Output vectors from different heads are concatenated (for each word)

$$\tilde{Y} = \begin{bmatrix} Y_1 \\ \vdots \\ Y_h \end{bmatrix}.$$

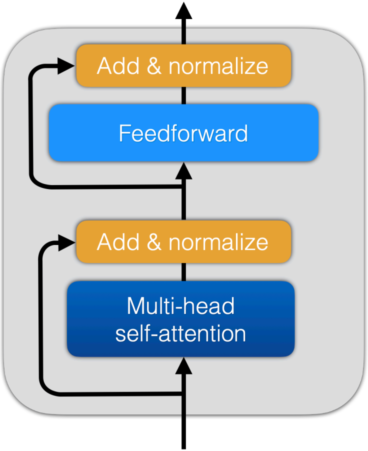
- Reduce dimensions back to d using a matrix W^o :

$$Y = W^o \tilde{Y}.$$

Example: $n = 3$, $d = 2$, $h = 3$



ADD & NORMALIZE



ADD & NORMALIZE

- Suppose the inputs are denoted X and Y, both size $d \times n$.

ADD & NORMALIZE

- Suppose the inputs are denoted X and Y, both size $d \times n$.

Add & normalization, out: Z

- Adding: $A = X + Y$.

ADD & NORMALIZE

- Suppose the inputs are denoted X and Y, both size $d \times n$.

Add & normalization, out: Z

- Adding: $A = X + Y$.
- Normalizing:

$$\mu = \frac{1}{d\,n} \sum_{i=1}^{d} \sum_{j=1}^{n} A_{ij}$$

$$\sigma^2 = \frac{1}{d\,n} \sum_{i=1}^{d} \sum_{j=1}^{n} (A_{ij} - \mu)^2$$

$$Z_{ij} = \frac{A_{ij} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ADD & NORMALIZE

- Suppose the inputs are denoted X and Y, both size $d \times n$.

Add & normalization, out: Z

- Adding: $A = X + Y$.
- Normalizing:

$$\mu = \frac{1}{d\,n} \sum_{i=1}^{d} \sum_{j=1}^{n} A_{ij}$$

$$\sigma^2 = \frac{1}{d\,n} \sum_{i=1}^{d} \sum_{j=1}^{n} (A_{ij} - \mu)^2$$

$$Z_{ij} = \frac{A_{ij} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

- Why add and normalize?

ADD & NORMALIZE

- Suppose the inputs are denoted X and Y, both size $d \times n$.

Add & normalization, out: Z

- Adding: $A = X + Y$.
- Normalizing:

$$\mu = \frac{1}{d\,n} \sum_{i=1}^{d} \sum_{j=1}^{n} A_{ij}$$

$$\sigma^2 = \frac{1}{d\,n} \sum_{i=1}^{d} \sum_{j=1}^{n} (A_{ij} - \mu)^2$$

$$Z_{ij} = \frac{A_{ij} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

- Why add and normalize?
- Adding $\Leftrightarrow$ residual connection:
  - yields stronger gradients,
  - helps remember positional encoding.

ADD & NORMALIZE

- Suppose the inputs are denoted X and Y, both size $d \times n$.

Add & normalization, out: Z

- Adding: $A = X + Y$.
- Normalizing:

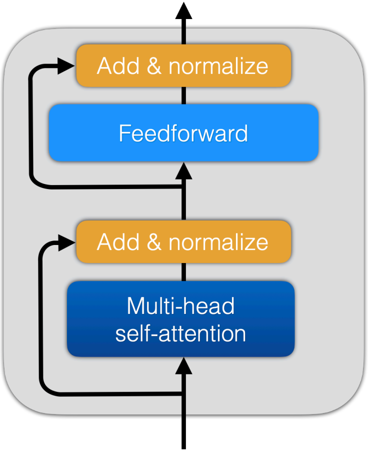
$$\mu = \frac{1}{d\,n} \sum_{i=1}^{d} \sum_{j=1}^{n} A_{ij}$$

$$\sigma^2 = \frac{1}{d\,n} \sum_{i=1}^{d} \sum_{j=1}^{n} (A_{ij} - \mu)^2$$

$$Z_{ij} = \frac{A_{ij} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

- Why add and normalize?
- Adding $\Leftrightarrow$ residual connection:
  - yields stronger gradients,
  - helps remember positional encoding.
- Normalization:
  - reduces covariate shift $\Rightarrow$ faster training,
  - centers embeddings around origin, which helps attention layers.

FEEDFORWARD NETWORKS

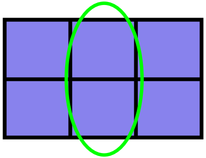


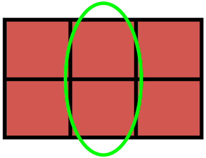
FEEDFORWARD NETWORKS

- Applies a fully connected network to each word embedding.

FEEDFORWARD NETWORKS

- Applies a fully connected network to each word embedding.

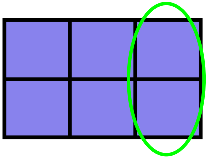
X 

Y 

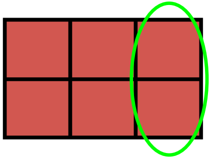
FEEDFORWARD NETWORKS

- Applies a fully connected network to each word embedding.
- Embeddings have the same in- and out-dimensions (true for all layers in the encoder).
- Weights are shared among all words.

X



Y

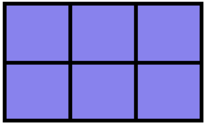


FEEDFORWARD NETWORKS

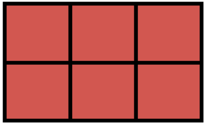
- Applies a fully connected network to each word embedding.
- Embeddings have the same in- and out-dimensions (true for all layers in the encoder).
- Weights are shared among all words.
- The specific form of the network is *linear layer* $\rightarrow$ *ReLU* $\rightarrow$ *linear layer*:

$$W_2 \max(0, W_1 x_i + b_1) + b_2.$$

X



Y



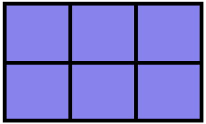
FEEDFORWARD NETWORKS

- Applies a fully connected network to each word embedding.
- Embeddings have the same in- and out-dimensions (true for all layers in the encoder).
- Weights are shared among all words.
- The specific form of the network is *linear layer* $\rightarrow$ *ReLU* $\rightarrow$ *linear layer*:

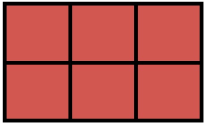
$$W_2 \max(0, W_1 x_i + b_1) + b_2.$$

- The operations correspond to two $1D$-convolution (fast implementation).

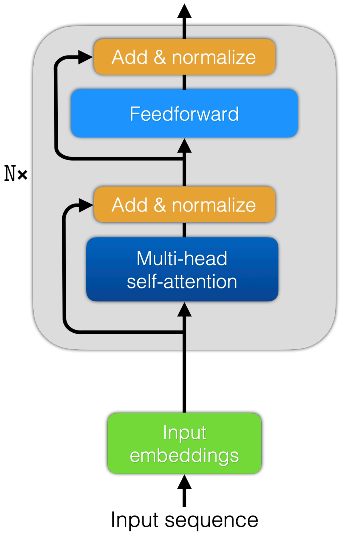
X



Y



SUMMARY



Add & normalize
Feedforward
N×
Add & normalize
Multi-head self-attention
Input embeddings
Input sequence

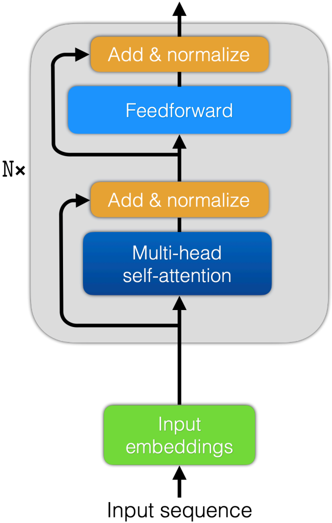
Encoder remarks

A series of videos on transformers

Lennart Svensson

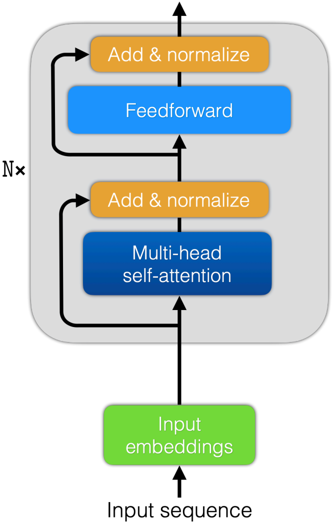
ENCODER IS A UNIVERSAL FUNCTION APPROXIMATOR

- The encoder contains two main components:
  - multi-head self-attention,
  - feedforward neural network (FFN).



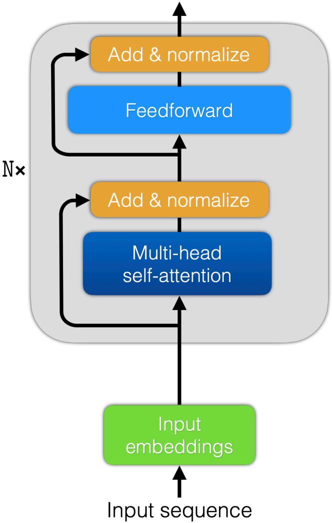
ENCODER IS A UNIVERSAL FUNCTION APPROXIMATOR

- The encoder contains two main components:
    - multi-head self-attention,
    - feedforward neural network (FFN).
- Multi-head self-attention is a "contextual mapping".



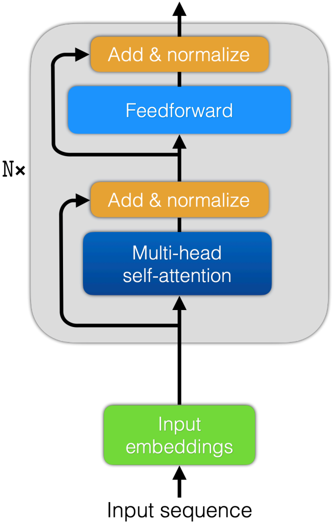
ENCODER IS A UNIVERSAL FUNCTION APPROXIMATOR

- The encoder contains two main components:
  - multi-head self-attention,
  - feedforward neural network (FFN).
- Multi-head self-attention is a "contextual mapping".
- FFN is applied token-wise.



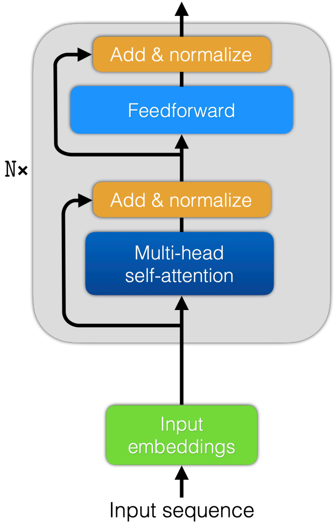
ENCODER IS A UNIVERSAL FUNCTION APPROXIMATOR

- The encoder contains two main components:
    - multi-head self-attention,
    - feedforward neural network (FFN).
- Multi-head self-attention is a "contextual mapping".
- FFN is applied token-wise.
- In spite all the weight-sharing, the encoder is a **universal function approximator**.



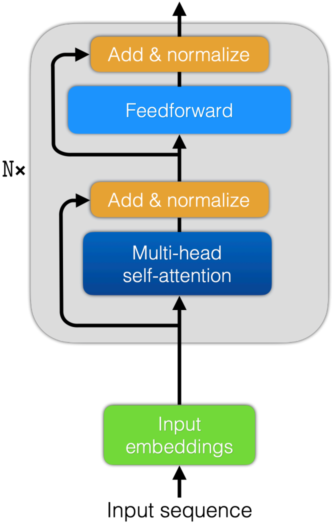
MULTI-HOP ATTENTION (1)

- First layer receives input embeddings.



MULTI-HOP ATTENTION (1)

- First layer receives input embeddings.
- Deeper layers perform self-attention on updated embeddings.



MULTI-HOP ATTENTION (2)

- Self-attention on updated embeddings $\Rightarrow$ “reason” in multiple steps?

MULTI-HOP ATTENTION (2)

- Self-attention on updated embeddings $\Rightarrow$ "reason" in multiple steps?

- Predict the next word?

  Karin has bought a **guitar**. When she came home, she left **it** in the kitchen and instead picked up some wine. When her brother entered the kitchen, he picked **it** up and started . . .

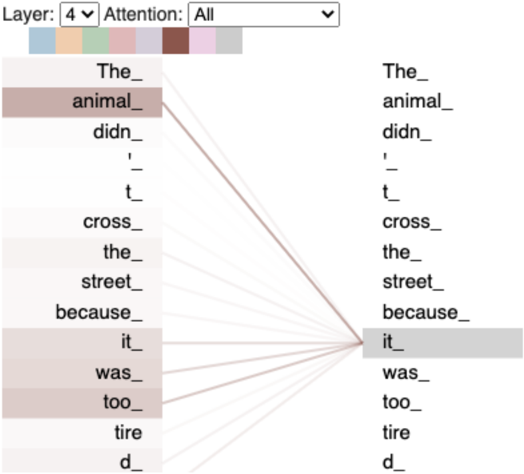
MULTI-HOP ATTENTION (2)

- Self-attention on updated embeddings $\Rightarrow$ “reason” in multiple steps?
  1. What did Karin leave in the kitchen?
  2. What did her brother pick up?
  3. What can you do with it?
- Predict the next word?

  Karin has bought a **guitar**. When she came home, she left **it** in the kitchen and instead picked up some wine. When her brother entered the kitchen, he picked **it** up and started . . .

VISUALIZATION OF ACTIVATIONS

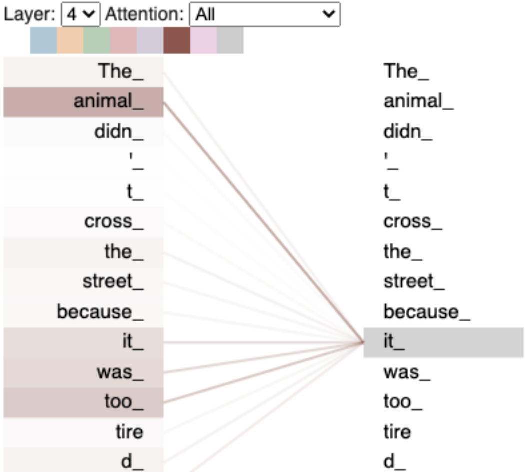
- Weights for the word “it”.



VISUALIZATION OF ACTIVATIONS

- Weights for the word “it”.

- Weights are specific for one attention head in layer 4 and one specific sentence.



POTENTIAL WEAKNESSES

POTENTIAL WEAKNESSES

- Complexity is $O(n^2)$, where n is sequence length.
- Recall that for all $i, j \in \{1, 2, \ldots, n\}$:

$$Z_{ji} = k_j^\top q_i / \sqrt{d}.$$

POTENTIAL WEAKNESSES

- Complexity is $O(n^2)$, where n is sequence length.
- Recall that for all $i, j \in \{1, 2, \ldots, n\}$:

$$Z_{ji} = k_j^\top q_i / \sqrt{d}.$$

- Can be difficult to train.

POTENTIAL WEAKNESSES

- Complexity is $O(n^2)$, where n is sequence length.
- Recall that for all $i, j \in \{1, 2, \ldots, n\}$:

$$Z_{ji} = k_j^\top q_i / \sqrt{d}.$$

- Can be difficult to train.

- Many heads are unimportant and can be pruned with little impact on performance.

Transformers vs CNNs and RNNs

A series of videos on transformers

Lennart Svensson

CONVOLUTIONAL NEURAL NETWORKS (CNNs)

Commonly used for images, data sequences and more.

A toy example

Filter:

-1	0	1
-1	0	1
-1	0	1

Input:

1	0	1	0	-1
0	1	1	0	-1
-1	0	1	1	0
-1	0	1	0	0

Output:

3	0	-5
5	0	-4

CONVOLUTIONAL NEURAL NETWORKS (CNNs)

Commonly used for images, data sequences and more.

A few key properties

- **Local filters** (and thus features).

A toy example

Filter:

-1	0	1
-1	0	1
-1	0	1

Input:

1	0	1	0	-1
0	1	1	0	-1
-1	0	1	1	0
-1	0	1	0	0

Output:

3	0	-5
5	0	-4

CONVOLUTIONAL NEURAL NETWORKS (CNNs)

Commonly used for images, data sequences and more.

A few key properties

- **Local filters** (and thus features).
- Weights depend on **relative position**.

A toy example

Filter:

-1	0	1
-1	0	1
-1	0	1

Input:

1	0	1	0	-1
0	1	1	0	-1
-1	0	1	1	0
-1	0	1	0	0

Output:

3	0	-5
5	0	-4

CONVOLUTIONAL NEURAL NETWORKS (CNNs)

Commonly used for images, data sequences and more.

A few key properties

- **Local filters** (and thus features).
- Weights depend on **relative position**.

- Other properties: Weights are shared across entire image, pooling layers, etc.

A toy example

Filter:

-1	0	1
-1	0	1
-1	0	1

Input:

1	0	1	0	-1
0	1	1	0	-1
-1	0	1	1	0
-1	0	1	0	0

Output:

3	0	-5
5	0	-4

CONVOLUTIONAL NEURAL NETWORKS (CNNs)

Commonly used for images, data sequences and more.

A few key properties

- **Local filters** (and thus features).
- Weights depend on **relative position**.

- Other properties: Weights are shared across entire image, pooling layers, etc.

Text examples: This book was written by Johan.
Johan wrote this book.

A toy example

Filter:

-1	0	1
-1	0	1
-1	0	1

Input:

1	0	1	0	-1
0	1	1	0	-1
-1	0	1	1	0
-1	0	1	0	0

Output:

3	0	-5
5	0	-4

CONVOLUTIONAL NEURAL NETWORKS (CNNs)

Commonly used for images, data sequences and more.

A few key properties

- **Local filters** (and thus features).
- Weights depend on **relative position**.

- Other properties: Weights are shared across entire image, pooling layers, etc.

A toy example

Filter: $\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$

Input: $\begin{array}{|c|c|c|c|c|}\hline 1 & 0 & 1 & 0 & -1 \\ \hline 0 & 1 & 1 & 0 & -1 \\ \hline -1 & 0 & 1 & 1 & 0 \\ \hline -1 & 0 & 1 & 0 & 0 \\ \hline \end{array}$

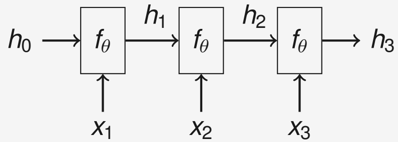
Output: $\begin{array}{|c|c|c|}\hline 3 & 0 & -5 \\ \hline 5 & 0 & -4 \\ \hline \end{array}$

Text examples: This book, which is about a girl from India, was written by Johan.

RECURRENT NEURAL NETWORKS (RNNs)

Commonly used to model sequences: dynamic models, language and more.

Basic RNN structure



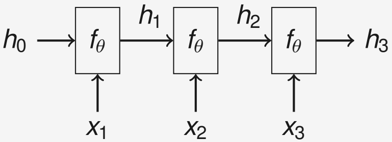
RECURRENT NEURAL NETWORKS (RNNs)

Commonly used to model sequences: dynamic models, language and more.

A few key properties

- States only depend on earlier input.

Basic RNN structure



Text examples: She went to the bank of the river.

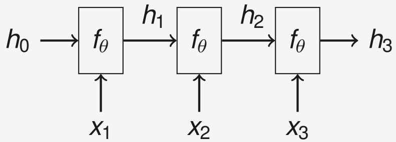
RECURRENT NEURAL NETWORKS (RNNs)

Commonly used to model sequences:
dynamic models, language and more.

A few key properties

- States only depend on earlier input.
- States computed in order.

Basic RNN structure



Text examples: She went to the bank of the river.
This book, which is about a girl from India, was written by Johan.

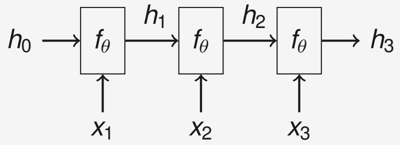
RECURRENT NEURAL NETWORKS (RNNs)

Commonly used to model sequences: dynamic models, language and more.

A few key properties

- States only depend on earlier input.
- States computed in order.

Basic RNN structure



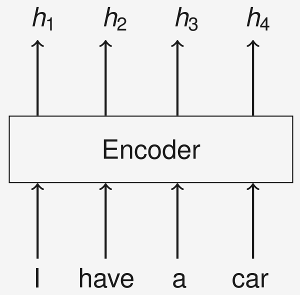
- Other properties: Weights are shared, can handle variable length input-output, etc.

Text examples: She went to the bank of the river.
This book, which is about a girl from India, was written by Johan.

TRANSFORMER ENCODER

Maps sets to sets!

Encoding each word



TRANSFORMER ENCODER

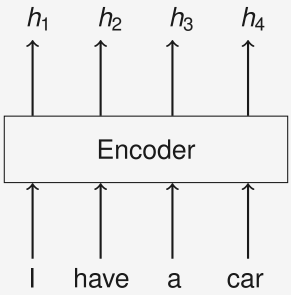
Maps sets to sets!

A few key properties

- Order is “ignored”

  (but position is encoded in state).

Encoding each word



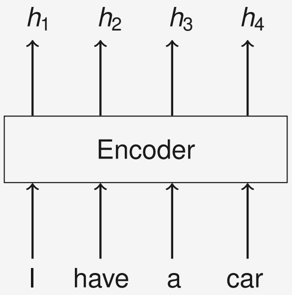
TRANSFORMER ENCODER

Maps sets to sets!

A few key properties

- Order is "ignored"
  (but position is encoded in state).
- Feature vectors depend on entire sequence.

Encoding each word



Text example: She went to the bank of the river.

TRANSFORMER ENCODER

Maps sets to sets!

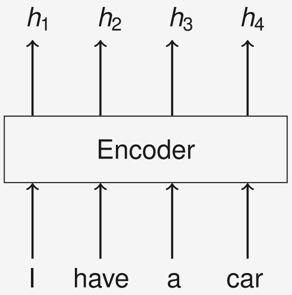
A few key properties

- Order is "ignored"
  (but position is encoded in state).
- Feature vectors depend on entire sequence.

- Other properties: Same number of input and output vectors, deep self-attention architecture, etc.

Text example: She went to the bank of the river.

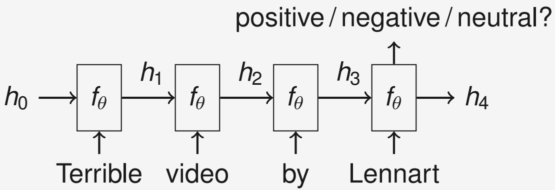
Encoding each word



VANISHING GRADIENTS VS TRANSFORMERS

RNN: example and properties

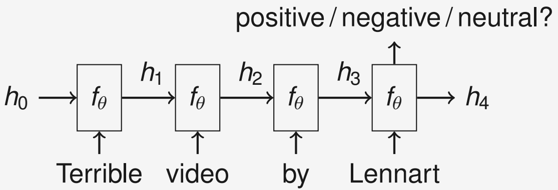
- may forget the past



VANISHING GRADIENTS VS TRANSFORMERS

RNN: example and properties

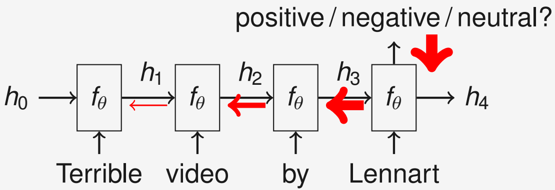
- may forget the past
- may suffer from vanishing gradients.



VANISHING GRADIENTS VS TRANSFORMERS

RNN: example and properties

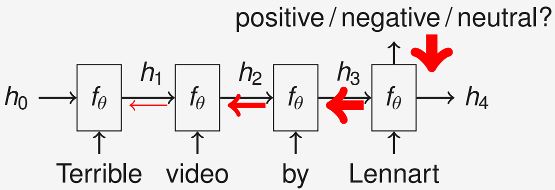
- may forget the past
- may suffer from vanishing gradients.



VANISHING GRADIENTS VS TRANSFORMERS

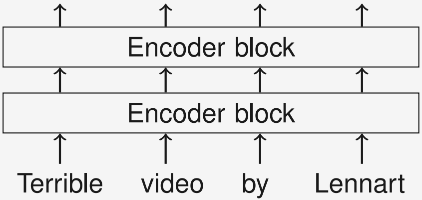
RNN: example and properties

- may forget the past
- may suffer from vanishing gradients.



Transformer encoder: example and properties

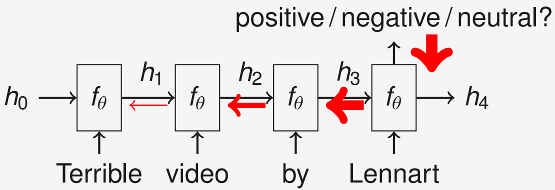
- maintains one vector per word,



VANISHING GRADIENTS VS TRANSFORMERS

RNN: example and properties

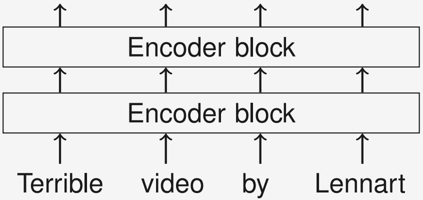
- may forget the past
- may suffer from vanishing gradients.



Transformer encoder: example and properties

- maintains one vector per word,
- directly links output to all input words

$$y_4 = \sum_j \underbrace{v_j}_{W_V x_j} W_{j4}.$$

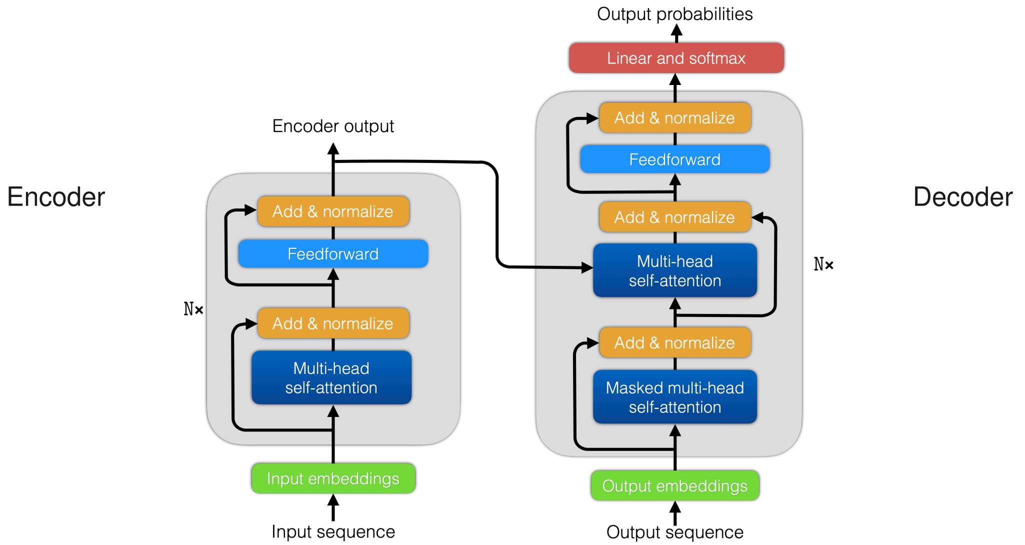


Decoder: testing and training

A series of videos on transformers

Lennart Svensson

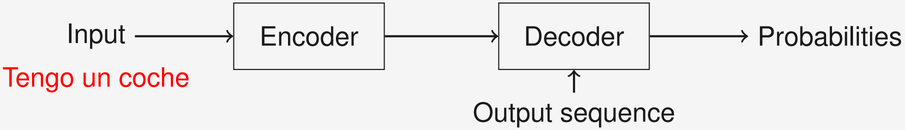
TRANSFORMER OVERVIEW



USING THE DECODER (TESTING)

Translating Spanish to English

Target sentence: I have a car.



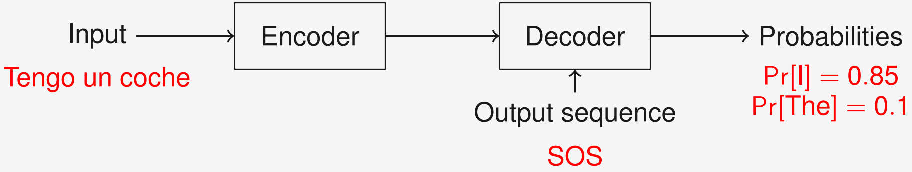
Repeat until end of sequence:

1) Input translated words.
2) Compute next word probabilities.
3) Pick (sample?) next word.

USING THE DECODER (TESTING)

Translating Spanish to English

Target sentence: I have a car.



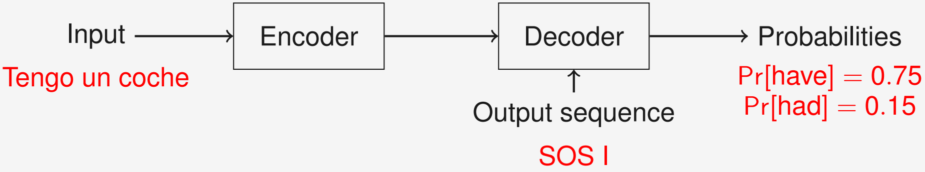
Repeat until end of sequence:

1) Input translated words.
2) Compute next word probabilities.
3) Pick (sample?) next word.

USING THE DECODER (TESTING)

Translating Spanish to English

Target sentence: I have a car.



Repeat until end of sequence:

1) Input translated words.
2) Compute next word probabilities.
3) Pick (sample?) next word.

USING THE DECODER (TESTING)

Translating Spanish to English

Target sentence: I have a car.

Input → Encoder → Decoder → Probabilities

Tengo un coche

Output sequence

SOS I have

$\Pr[a] = 0.9$

$\Pr[an] = 0.05$

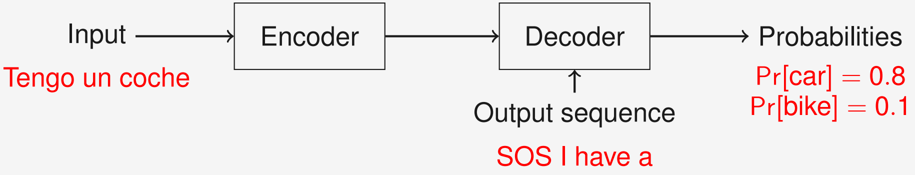
Repeat until end of sequence:

1) Input translated words.
2) Compute next word probabilities.
3) Pick (sample?) next word.

USING THE DECODER (TESTING)

Translating Spanish to English

Target sentence: I have a car.



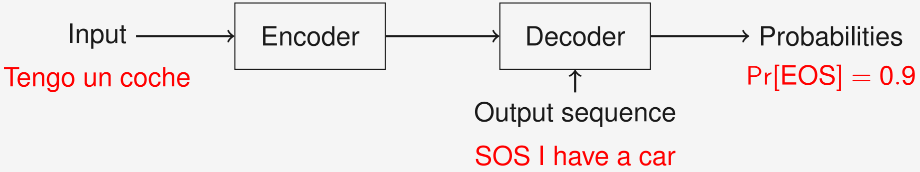
Repeat until end of sequence:

1) Input translated words.
2) Compute next word probabilities.
3) Pick (sample?) next word.

USING THE DECODER (TESTING)

Translating Spanish to English

Target sentence: I have a car.



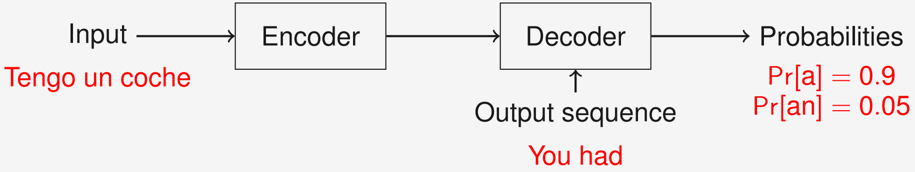
Repeat until end of sequence:

1) Input translated words.
2) Compute next word probabilities.
3) Pick (sample?) next word.

USING THE DECODER (TESTING)

Translating Spanish to English

Target sentence: I have a car.



Repeat until end of sequence:

1) Input translated words.
2) Compute next word probabilities.
3) Pick (sample?) next word.

- **Note:** translation is feeded back into decoder. It might be incorrect!

TRAINING THE DECODER

Use training data pairs (original sequence, target translation):

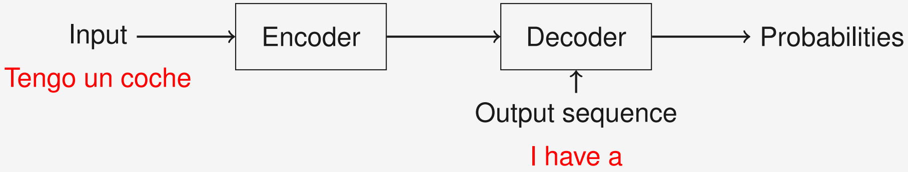
TRAINING THE DECODER

Use training data pairs (original sequence, target translation):

1) Input original sentence and part of target sentence.

Translating Spanish to English

Target sentence: I have a car.



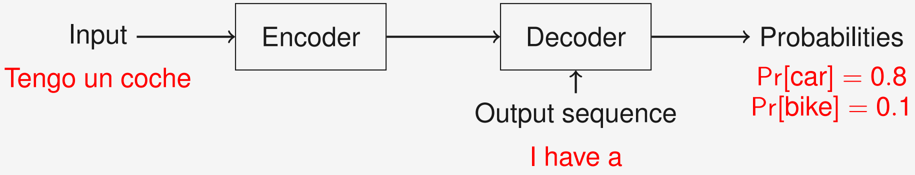
TRAINING THE DECODER

Use training data pairs (original sequence, target translation):

1) Input original sentence and part of target sentence.
2) Compute next word probabilities.

Translating Spanish to English

Target sentence: I have a car.



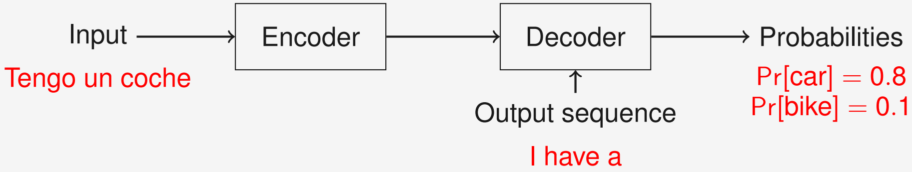
TRAINING THE DECODER

Use training data pairs (original sequence, target translation):

1) Input original sentence and part of target sentence.
2) Compute next word probabilities.
3) Minimize cross-entropy loss: maximize probability of target word.

Translating Spanish to English

Target sentence: I have a car.



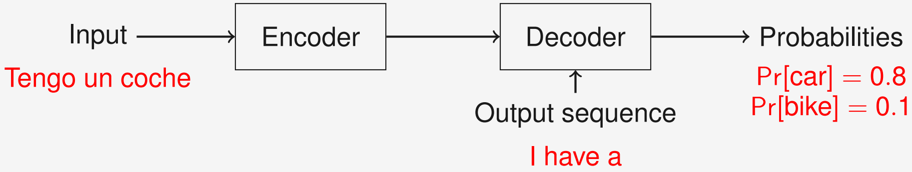
TRAINING THE DECODER

Use training data pairs (original sequence, target translation):

1) Input original sentence and part of target sentence.
2) Compute next word probabilities.
3) Minimize cross-entropy loss: maximize probability of target word.

Translating Spanish to English

Target sentence: I have a car.



- **Note 1:** translation is determined by target, **teacher forcing.**

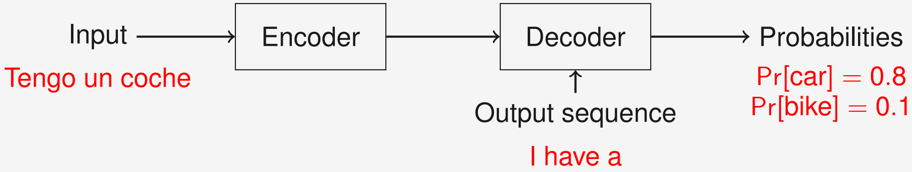
TRAINING THE DECODER

Use training data pairs (original sequence, target translation):

1) Input original sentence and part of target sentence.
2) Compute next word probabilities.
3) Minimize cross-entropy loss: maximize probability of target word.

Translating Spanish to English

Target sentence: I have a car.



- **Note 1:** translation is determined by target, **teacher forcing.**
- **Note 2:** predicts probabilities of one word at a time (inefficient?).

EMPIRICAL RISK MINIMIZATION

- Training data

$$\left(x^{(j)}_{1:n_x^{(j)}}, y^{(j)}_{1:n_y^{(j)}}\right), \quad j = 1, \ldots, m,$$

where

$$x^{(j)}_{1:n_x^{(j)}} = (x^{(j)}_1, x^{(j)}_2, \ldots, x^{(j)}_{n_x^{(j)}})\text{: original sequence,}$$
$$y^{(j)}_{1:n_y^{(j)}} = (y^{(j)}_1, y^{(j)}_2, \ldots, y^{(j)}_{n_y^{(j)}})\text{: translation.}$$

EMPIRICAL RISK MINIMIZATION

- Training data

$$\left(x^{(j)}_{1:n_x^{(j)}}, y^{(j)}_{1:n_y^{(j)}}\right), \quad j = 1, \dots, m,$$

where

$$x^{(j)}_{1:n_x^{(j)}} = (x^{(j)}_1, x^{(j)}_2, \dots, x^{(j)}_{n_x^{(j)}})\text{: original sequence,}$$
$$y^{(j)}_{1:n_y^{(j)}} = (y^{(j)}_1, y^{(j)}_2, \dots, y^{(j)}_{n_y^{(j)}})\text{: translation.}$$

- Select network parameters that minimize total loss across training data

$$\sum_{j=1}^{m} \sum_{i=2}^{n_y^{(j)}} -\log \Pr\left[y^{(j)}_i \middle| y^{(j)}_{1:i-1}, x^{(j)}_{1:n_x^{(j)}}\right].$$

EMPIRICAL RISK MINIMIZATION

- Training data

$$\left(x^{(j)}_{1:n_x^{(j)}}, y^{(j)}_{1:n_y^{(j)}}\right), \quad j = 1, \ldots, m,$$

  where

$$x^{(j)}_{1:n_x^{(j)}} = (x^{(j)}_1, x^{(j)}_2, \ldots, x^{(j)}_{n_x^{(j)}})\text{: original sequence,}$$
$$y^{(j)}_{1:n_y^{(j)}} = (y^{(j)}_1, y^{(j)}_2, \ldots, y^{(j)}_{n_y^{(j)}})\text{: translation.}$$

- Select network parameters that minimize total loss across training data

$$\sum_{j=1}^{m} \underbrace{\sum_{i=2}^{n_y^{(j)}}}_{\textbf{In parallel?}} -\log \Pr\left[y_i^{(j)} \middle| y^{(j)}_{1:i-1}, x^{(j)}_{1:n_x^{(j)}}\right].$$

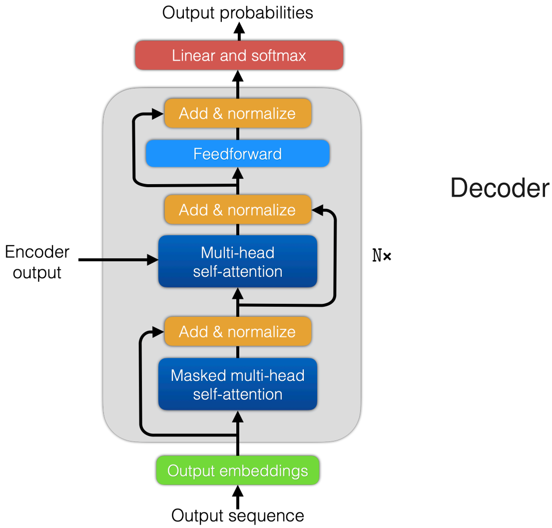
Decoder: masked self-attention

A series of videos on transformers

Lennart Svensson

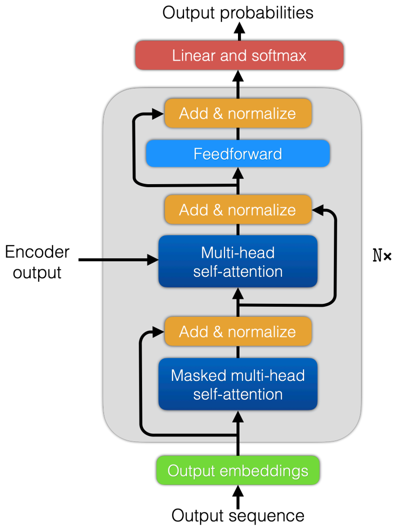
TRANSFORMER OVERVIEW

- The decoder stacks *N* decoder blocks.



TRANSFORMER OVERVIEW

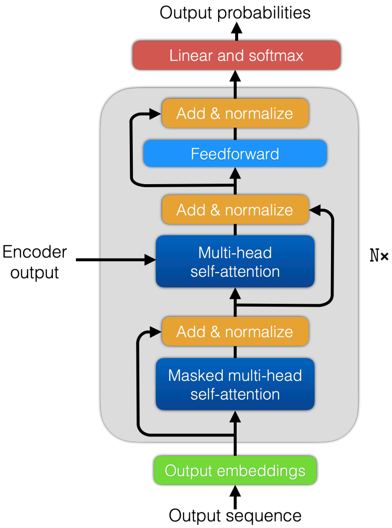
- The decoder stacks *N* decoder blocks.
- Each decoder block maintains shape (#vectors, vector length).



Decoder

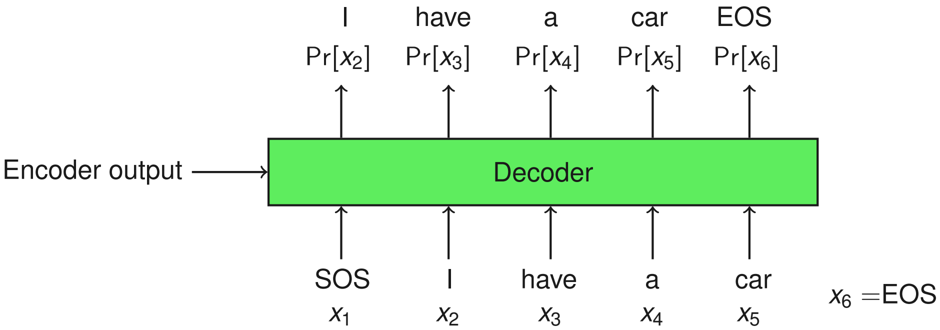
TRANSFORMER OVERVIEW

- The decoder stacks N decoder blocks.
- Each decoder block maintains shape (#vectors, vector length).
- We have #vectors$= n_D$, the length of output sequence.



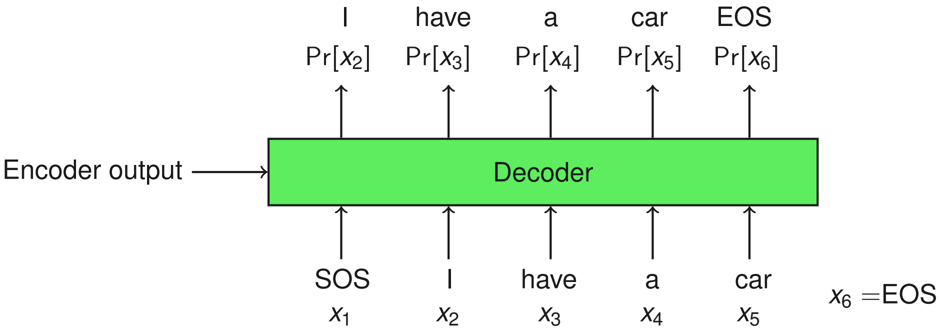
Decoder

PARALLELIZED PREDICTIONS DURING TRAINING



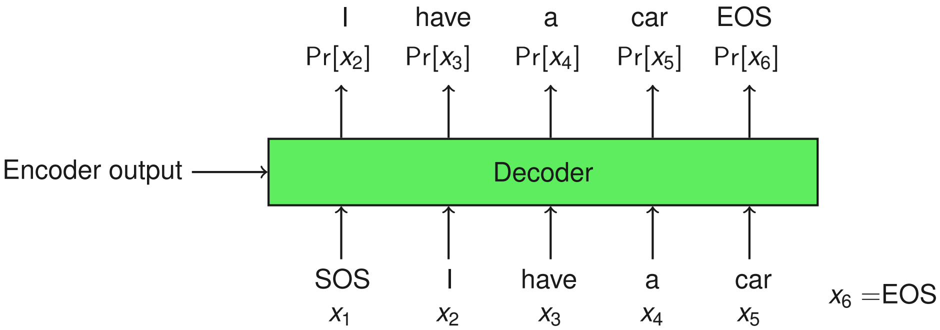
- Compute all next word probabilities in parallel?
  - Feed entire target sequence into decoder.

PARALLELIZED PREDICTIONS DURING TRAINING



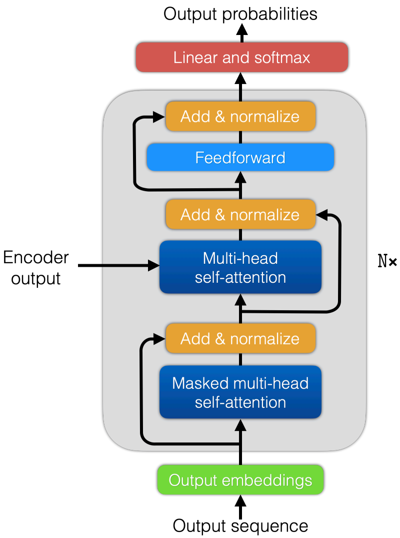
- Compute all next word probabilities in parallel?
  - Feed entire target sequence into decoder.
  - Only possible during training.

PARALLELIZED PREDICTIONS DURING TRAINING



- Compute all next word probabilities in parallel?
    - Feed entire target sequence into decoder.
    - Only possible during training.
    - Predictions may not use future input.

MASKED SELF-ATTENTION (1)

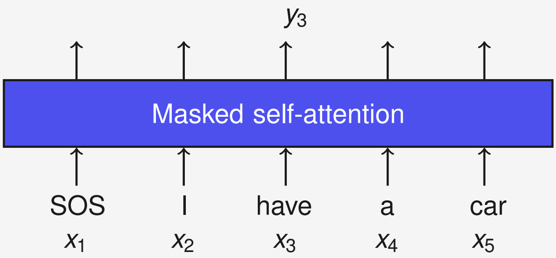


Decoder

MASKED SELF-ATTENTION (1)

Example: computing y_3

- y_3 should only depend on x_1, x_2, x_3.

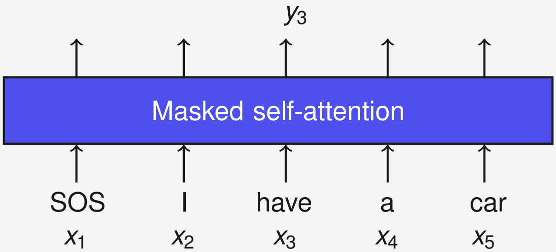


MASKED SELF-ATTENTION (1)

Example: computing y_3

- y_3 should only depend on x_1, x_2, x_3.
- Compute queries, keys and Z:

$$Z_{13} = \frac{k_1^\top q_3}{\sqrt{d}}, \ldots, Z_{53} = \frac{k_5^\top q_3}{\sqrt{d}}.$$

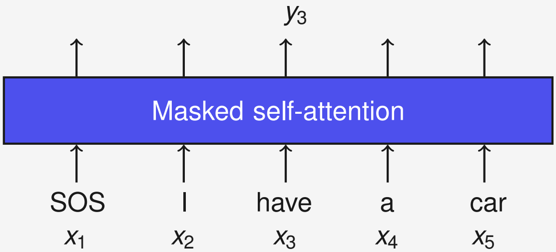


MASKED SELF-ATTENTION (1)

Example: computing y_3

- y_3 should only depend on x_1, x_2, x_3.
- Compute queries, keys and Z:

$$Z_{13} = \frac{k_1^T q_3}{\sqrt{d}}, \dots, Z_{53} = \frac{k_5^T q_3}{\sqrt{d}}.$$



- “Masked” weights:

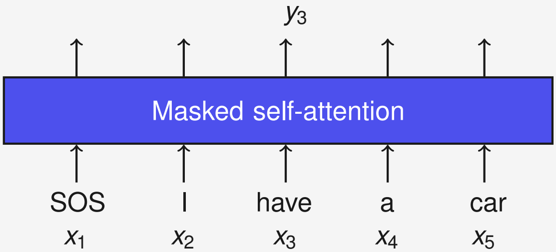
$$\begin{bmatrix} \tilde{W}_{13} \\ \tilde{W}_{23} \\ \tilde{W}_{33} \\ \tilde{W}_{43} \\ \tilde{W}_{53} \end{bmatrix} = \begin{bmatrix} \exp(Z_{13}) \\ \exp(Z_{23}) \\ \exp(Z_{33}) \\ 0 \\ 0 \end{bmatrix}$$

MASKED SELF-ATTENTION (1)

Example: computing y_3

- y_3 should only depend on x_1, x_2, x_3.
- Compute queries, keys and Z:

$$Z_{13} = \frac{k_1^T q_3}{\sqrt{d}}, \ldots, Z_{53} = \frac{k_5^T q_3}{\sqrt{d}}.$$



- "Masked" weights:

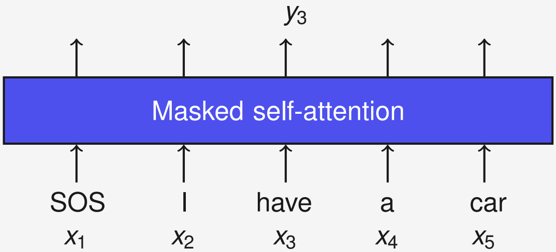
$$\begin{bmatrix} \tilde{W}_{13} \\ \tilde{W}_{23} \\ \tilde{W}_{33} \\ \tilde{W}_{43} \\ \tilde{W}_{53} \end{bmatrix} = \begin{bmatrix} \exp(Z_{13}) \\ \exp(Z_{23}) \\ \exp(Z_{33}) \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} W_{13} \\ W_{23} \\ W_{33} \\ W_{43} \\ W_{53} \end{bmatrix} = \begin{bmatrix} \tilde{W}_{13} \\ \tilde{W}_{23} \\ \tilde{W}_{33} \\ 0 \\ 0 \end{bmatrix} \frac{1}{\tilde{W}_{13} + \tilde{W}_{23} + \tilde{W}_{33}}$$

MASKED SELF-ATTENTION (1)

Example: computing y_3

- y_3 should only depend on x_1, x_2, x_3.
- Compute queries, keys and Z:

$$Z_{13} = \frac{k_1^T q_3}{\sqrt{d}}, \ldots, Z_{53} = \frac{k_5^T q_3}{\sqrt{d}}.$$



- "Masked" weights:

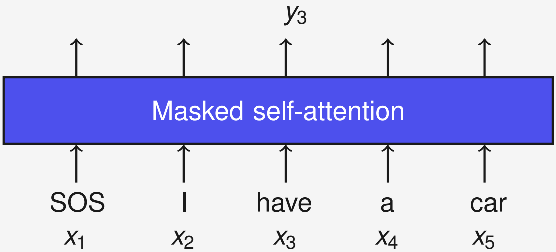
$$\begin{bmatrix} \tilde{W}_{13} \\ \tilde{W}_{23} \\ \tilde{W}_{33} \\ \tilde{W}_{43} \\ \tilde{W}_{53} \end{bmatrix} = \begin{bmatrix} \exp(Z_{13}) \\ \exp(Z_{23}) \\ \exp(Z_{33}) \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} W_{13} \\ W_{23} \\ W_{33} \\ W_{43} \\ W_{53} \end{bmatrix} = \begin{bmatrix} \mathrm{softmax}(Z_{13}, Z_{23}, Z_{33}) \\ \\ 0 \\ 0 \end{bmatrix}$$

MASKED SELF-ATTENTION (1)

Example: computing y_3

- y_3 should only depend on x_1, x_2, x_3.
- Compute queries, keys and Z:

$$Z_{13} = \frac{k_1^T q_3}{\sqrt{d}}, \ldots, Z_{53} = \frac{k_5^T q_3}{\sqrt{d}}.$$



- "Masked" weights:

$$\begin{bmatrix} \tilde{W}_{13} \\ \tilde{W}_{23} \\ \tilde{W}_{33} \\ \tilde{W}_{43} \\ \tilde{W}_{53} \end{bmatrix} = \begin{bmatrix} \exp(Z_{13}) \\ \exp(Z_{23}) \\ \exp(Z_{33}) \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} W_{13} \\ W_{23} \\ W_{33} \\ W_{43} \\ W_{53} \end{bmatrix} = \begin{bmatrix} \mathrm{softmax}(Z_{13}, Z_{23}, Z_{33}) \\ \\ \\ 0 \\ 0 \end{bmatrix} \Rightarrow y_3 = \sum_{i=1}^{5} v_i W_{i3}$$

MASKED SELF-ATTENTION (2)

- Calculations without for-loops:

$$
\begin{aligned}
\text{Query:} \qquad & Q = W_Q X, \\
\text{Keys:} \qquad & K = W_K X, \\
\text{Values:} \qquad & V = W_V X \\
& Z = K^{\mathsf{T}} Q / \sqrt{d}.
\end{aligned}
$$

MASKED SELF-ATTENTION (2)

- Calculations without for-loops:

$$\begin{aligned} \text{Query:} \qquad & Q = W_Q X, \\ \text{Keys:} \qquad & K = W_K X, \\ \text{Values:} \qquad & V = W_V X \\ & Z = K^\top Q / \sqrt{d}. \end{aligned}$$

- Unnormalized weights are **masked**:

$$\tilde{W} = \begin{bmatrix} \exp(Z_{11}) & \exp(Z_{12}) & \exp(Z_{13}) & \exp(Z_{14}) & \exp(Z_{15}) & \dots \\ 0 & \exp(Z_{22}) & \exp(Z_{23}) & \exp(Z_{14}) & \exp(Z_{25}) & \dots \\ 0 & 0 & \exp(Z_{33}) & \exp(Z_{34}) & \exp(Z_{35}) & \dots \\ 0 & 0 & 0 & \exp(Z_{44}) & \exp(Z_{45}) & \dots \\ 0 & 0 & 0 & 0 & \exp(Z_{55}) & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

MASKED SELF-ATTENTION (2)

- Calculations without for-loops:

$$\begin{aligned} \text{Query:} \quad & Q = W_Q X, \\ \text{Keys:} \quad & K = W_K X, \\ \text{Values:} \quad & V = W_V X \\ & Z = K^\top Q / \sqrt{d}. \end{aligned}$$

- Weights using columnwise **masked** softmax (sm):

$$W = \begin{bmatrix} 1 & \mathrm{sm}_1(Z_{12}, Z_{22}) & \mathrm{sm}_1(Z_{13}, Z_{23}, Z_{33}) & \mathrm{sm}_1(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \cdots \\ 0 & \mathrm{sm}_2(Z_{12}, Z_{22}) & \mathrm{sm}_2(Z_{13}, Z_{23}, Z_{33}) & \mathrm{sm}_2(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \cdots \\ 0 & 0 & \mathrm{sm}_3(Z_{13}, Z_{23}, Z_{33}) & \mathrm{sm}_3(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \cdots \\ 0 & 0 & 0 & \mathrm{sm}_4(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

MASKED SELF-ATTENTION (2)

- Calculations without for-loops:

$$\begin{aligned} \text{Query:} \qquad & Q = W_Q X, \\ \text{Keys:} \qquad & K = W_K X, \\ \text{Values:} \qquad & V = W_V X \\ & Z = K^{\mathsf{T}} Q / \sqrt{d}. \end{aligned}$$

- Weights using columnwise **masked** softmax (sm):

$$W = \begin{bmatrix} 1 & \mathrm{sm}_1(Z_{12}, Z_{22}) & \mathrm{sm}_1(Z_{13}, Z_{23}, Z_{33}) & \mathrm{sm}_1(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \dots \\ 0 & \mathrm{sm}_2(Z_{12}, Z_{22}) & \mathrm{sm}_2(Z_{13}, Z_{23}, Z_{33}) & \mathrm{sm}_2(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \dots \\ 0 & 0 & \mathrm{sm}_3(Z_{13}, Z_{23}, Z_{33}) & \mathrm{sm}_3(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \dots \\ 0 & 0 & 0 & \mathrm{sm}_4(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

- New embeddings using weighted averages:

$$Y = VW \qquad \Rightarrow \qquad y_i = v_1 W_{1i} + v_2 W_{2i} + \dots + v_i W_{ii}.$$

MASKED SELF-ATTENTION (2)

- Calculations without for-loops:

$$\begin{aligned}
\text{Query:} \qquad & Q = W_Q X, \\
\text{Keys:} \qquad & K = W_K X, \\
\text{Values:} \qquad & V = W_V X \\
& Z = K^\top Q/\sqrt{d}.
\end{aligned}$$

- Weights using columnwise **masked** softmax (sm):

$$W = \begin{bmatrix}
1 & \text{sm}_1(Z_{12}, Z_{22}) & \text{sm}_1(Z_{13}, Z_{23}, Z_{33}) & \text{sm}_1(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \dots \\
0 & \text{sm}_2(Z_{12}, Z_{22}) & \text{sm}_2(Z_{13}, Z_{23}, Z_{33}) & \text{sm}_2(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \dots \\
0 & 0 & \text{sm}_3(Z_{13}, Z_{23}, Z_{33}) & \text{sm}_3(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \dots \\
0 & 0 & 0 & \text{sm}_4(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \dots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{bmatrix}$$

- New embeddings using weighted averages:

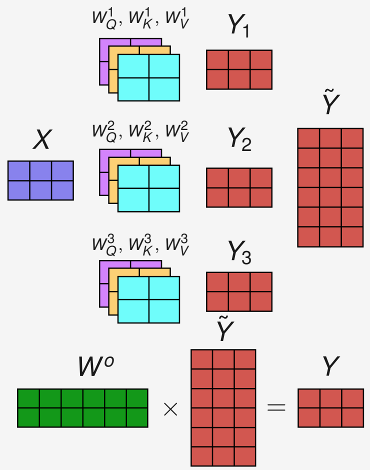
$$Y = VW \qquad \Rightarrow \qquad y_i = v_1 W_{1i} + v_2 W_{2i} + \cdots + v_i W_{ii}.$$

- Order matters! Not a mapping from one set to another.

MASKED MULTI-HEAD SELF-ATTENTION

- h parallel **masked** self-attention blocks/heads.

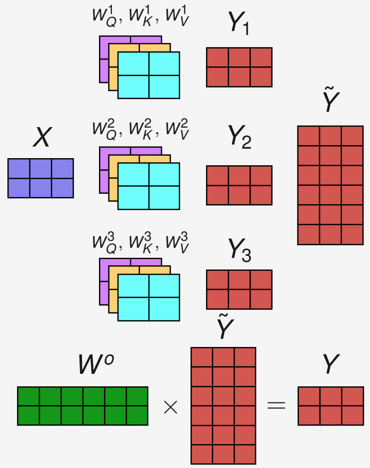
Example: $n = 3$, $d = 2$, $h = 3$



MASKED MULTI-HEAD SELF-ATTENTION

- h parallel **masked** self-attention blocks/heads.
- Overall structure is identical to multi-head attention.

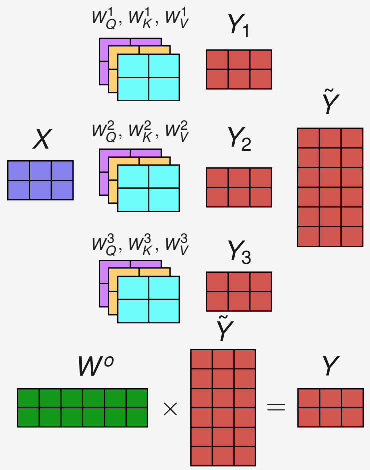
Example: $n = 3$, $d = 2$, $h = 3$



MASKED MULTI-HEAD SELF-ATTENTION

- h parallel **masked** self-attention blocks/heads.
- Overall structure is identical to multi-head attention.
- **Difference**: Y_i computed using masked softmax.

Example: $n = 3$, $d = 2$, $h = 3$



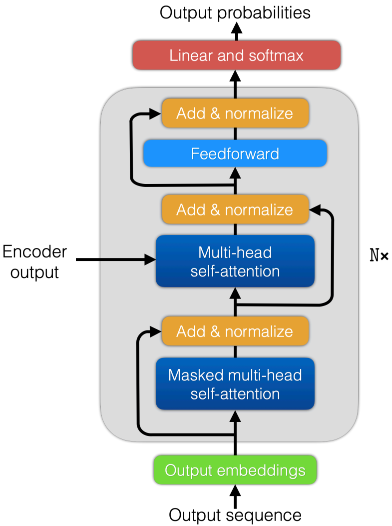
Decoder: encoder-decoder self-attention

A series of videos on transformers

Lennart Svensson

ENCODER-DECODER SELF-ATTENTION (1)

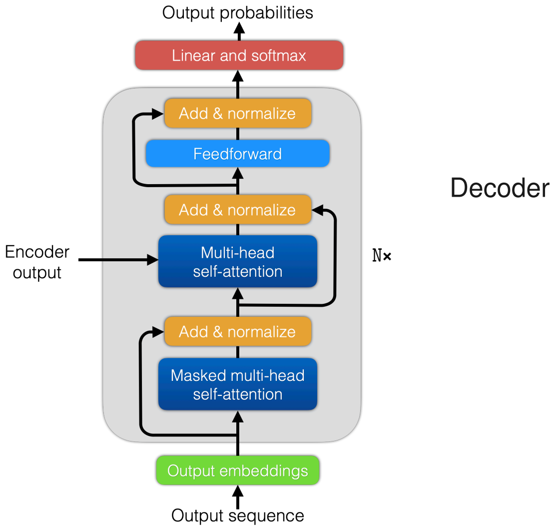
- Encoder-decoder self-attention takes two inputs: $X_D : d_D \times n_D$ and $X_E : d_E \times n_E$.



Decoder

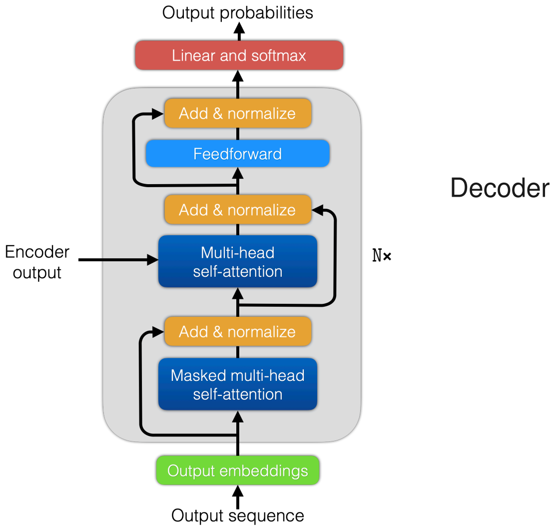
ENCODER-DECODER SELF-ATTENTION (1)

- Encoder-decoder self-attention takes two inputs: $X_D : d_D \times n_D$ and $X_E : d_E \times n_E$.
- Each decoder block maintains shape (#vectors, vector length).



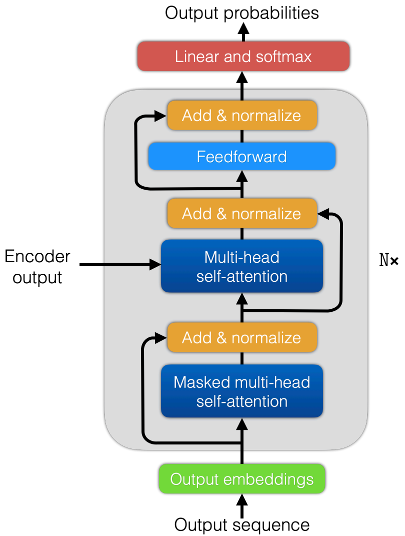
ENCODER-DECODER SELF-ATTENTION (1)

- Encoder-decoder self-attention takes two inputs: $X_D : d_D \times n_D$ and $X_E : d_E \times n_E$.
- Each decoder block maintains shape (#vectors, vector length).
- Output from encoder-decoder is $Y : d_D \times n_D$.



ENCODER-DECODER SELF-ATTENTION (2)

- Time to use encoder output!

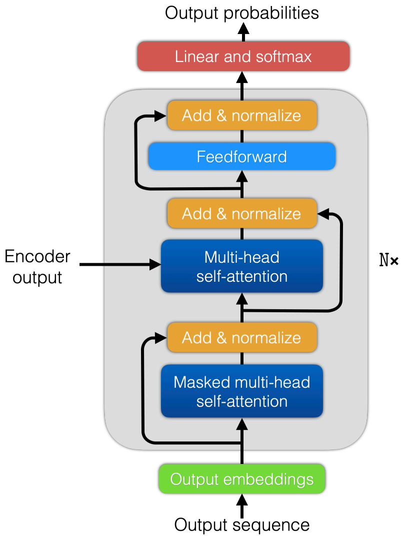


ENCODER-DECODER SELF-ATTENTION (2)

- Time to use encoder output!

Translating Spanish to English

- Input sentence: *Tengo un coche*.

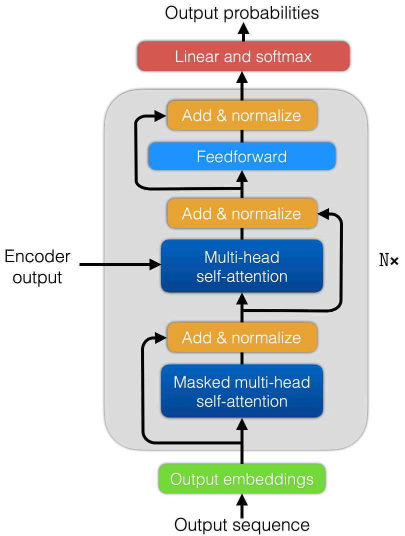


ENCODER-DECODER SELF-ATTENTION (2)

- Time to use encoder output!

Translating Spanish to English

- Input sentence: *Tengo un coche*.
- Encoder output: x_1^E, x_2^E, x_3^E.

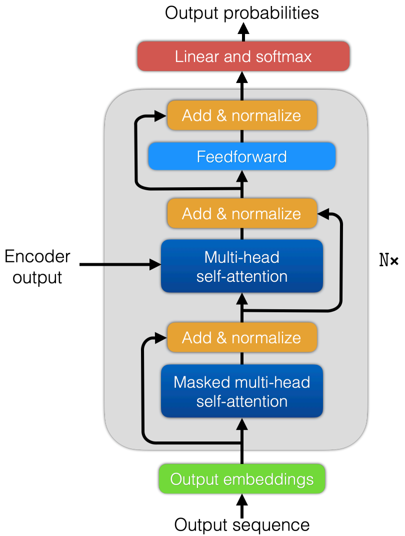


ENCODER-DECODER SELF-ATTENTION (2)

- Time to use encoder output!

Translating Spanish to English

- Input sentence: *Tengo un coche*.
- Encoder output: x_1^E, x_2^E, x_3^E.
- Finding y_4 for "encoder-decoder self-attention":

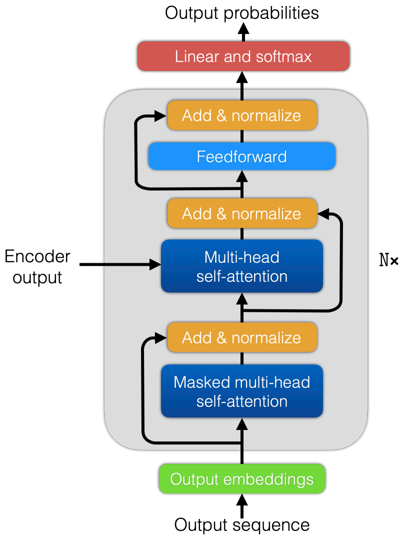


ENCODER-DECODER SELF-ATTENTION (2)

- Time to use encoder output!

Translating Spanish to English

- Input sentence: *Tengo un coche*.
- Encoder output: x_1^E, x_2^E, x_3^E.
- Finding y_4 for "encoder-decoder self-attention":
  - Query from x_4^D (decoder): $q_4 = W_Q x_4^D$.

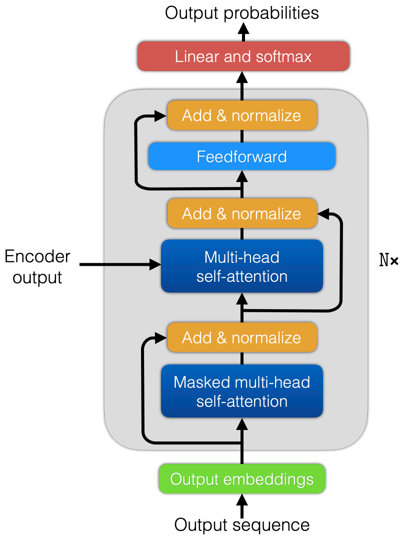


ENCODER-DECODER SELF-ATTENTION (2)

- Time to use encoder output!

Translating Spanish to English

- Input sentence: *Tengo un coche*.
- Encoder output: x_1^E, x_2^E, x_3^E.
- Finding y_4 for "encoder-decoder self-attention":
  - Query from x_4^D (decoder): $q_4 = W_Q x_4^D$.
  - Keys and values (encoder):
    $k_1 = W_K x_1^E, v_1 = W_V x_1^E, \ldots, v_3 = W_V x_3^E$.



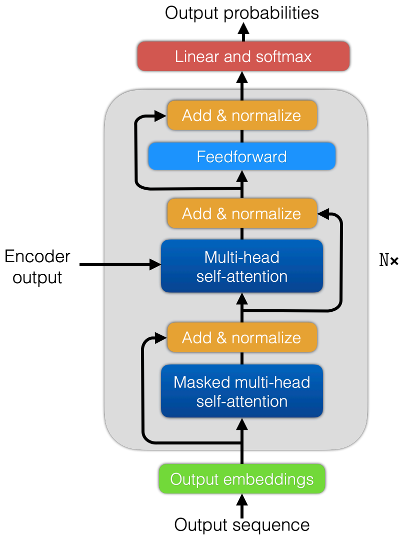
ENCODER-DECODER SELF-ATTENTION (2)

- Time to use encoder output!

Translating Spanish to English

- Input sentence: *Tengo un coche*.
- Encoder output: x_1^E, x_2^E, x_3^E.
- Finding y_4 for "encoder-decoder self-attention":
  - Query from x_4^D (decoder): $q_4 = W_Q x_4^D$.
  - Keys and values (encoder):
    $k_1 = W_K x_1^E, v_1 = W_V x_1^E, \ldots, v_3 = W_V x_3^E$.
  - Weights:
    $Z_{14} = k_1^T q_4/\sqrt{d}, Z_{24} = k_2^T q_4/\sqrt{d}, Z_{34} = k_3^T q_4/\sqrt{d}$

$$\left[W_{14},\ W_{24},\ W_{34}\right]^T = \mathrm{softmax}(\left[Z_{14},\ Z_{24},\ Z_{34}\right]^T).$$



ENCODER-DECODER SELF-ATTENTION (2)

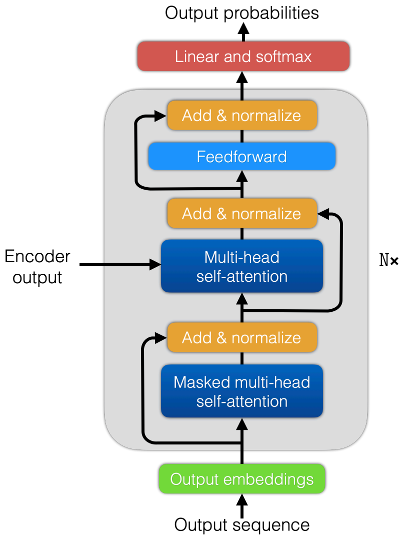
- Time to use encoder output!

Translating Spanish to English

- Input sentence: *Tengo un coche*.
- Encoder output: x_1^E, x_2^E, x_3^E.
- Finding y_4 for "encoder-decoder self-attention":
  - Query from x_4^D (decoder): $q_4 = W_Q x_4^D$.
  - Keys and values (encoder):
    $k_1 = W_K x_1^E, v_1 = W_V x_1^E, \dots, v_3 = W_V x_3^E$.
  - Weights:
    $Z_{14} = k_1^\mathrm{T} q_4/\sqrt{d}, Z_{24} = k_2^\mathrm{T} q_4/\sqrt{d}, Z_{34} = k_3^\mathrm{T} q_4/\sqrt{d}$

$$\begin{bmatrix} W_{14}, & W_{24}, & W_{34} \end{bmatrix}^\mathrm{T} = \mathrm{softmax}(\begin{bmatrix} Z_{14}, & Z_{24}, & Z_{34} \end{bmatrix}^\mathrm{T}).$$

  - Average: $y_4 = v_1 W_{14} + v_2 W_{24} + v_3 W_{34}$.



ENCODER-DECODER SELF-ATTENTION (3)

- Let us present equations in matrix form.

ENCODER-DECODER SELF-ATTENTION (3)

- Let us present equations in matrix form.
- Receives two inputs:

$$X_E = \begin{bmatrix} x_1^E & x_2^E & \dots & x_{n_E}^E \end{bmatrix}$$

$$X_D = \begin{bmatrix} x_1^D & x_2^D & \dots & x_{n_D}^D \end{bmatrix}.$$

ENCODER-DECODER SELF-ATTENTION (3)

- Let us present equations in matrix form.
- Receives two inputs:

$$X_E = \begin{bmatrix} x_1^E & x_2^E & \dots & x_{n_E}^E \end{bmatrix}$$

$$X_D = \begin{bmatrix} x_1^D & x_2^D & \dots & x_{n_D}^D \end{bmatrix}.$$

- **Queries** are computed using X_D:

$$Q = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_D} \end{bmatrix} = W_Q X_D.$$

ENCODER-DECODER SELF-ATTENTION (3)

- Let us present equations in matrix form.
- Receives two inputs:

$$X_E = \begin{bmatrix} x_1^E & x_2^E & \dots & x_{n_E}^E \end{bmatrix}$$

$$X_D = \begin{bmatrix} x_1^D & x_2^D & \dots & x_{n_D}^D \end{bmatrix}.$$

- **Queries** are computed using X_D:

$$Q = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_D} \end{bmatrix} = W_Q X_D.$$

- **Keys and values** are computed using X_E:

$$K = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_E} \end{bmatrix} = W_K X_E$$

$$V = \begin{bmatrix} v_1 & v_2 & \dots & v_{n_E} \end{bmatrix} = W_V X_E.$$

ENCODER-DECODER SELF-ATTENTION (3)

- Let us present equations in matrix form.
- Receives two inputs:

$$X_E = \begin{bmatrix} x_1^E & x_2^E & \dots & x_{n_E}^E \end{bmatrix}$$
$$X_D = \begin{bmatrix} x_1^D & x_2^D & \dots & x_{n_D}^D \end{bmatrix}.$$

- **Queries** are computed using X_D:

$$Q = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_D} \end{bmatrix} = W_Q X_D.$$

- **Keys and values** are computed using X_E:

$$K = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_E} \end{bmatrix} = W_K X_E$$
$$V = \begin{bmatrix} v_1 & v_2 & \dots & v_{n_E} \end{bmatrix} = W_V X_E.$$

  - Weights using columnwise softmax (no mask):

$$Z = K^\mathsf{T} Q/\sqrt{d}, \qquad W = \text{softmax}(Z).$$

ENCODER-DECODER SELF-ATTENTION (3)

- Let us present equations in matrix form.
- Receives two inputs:

$$X_E = \begin{bmatrix} x_1^E & x_2^E & \dots & x_{n_E}^E \end{bmatrix}$$

$$X_D = \begin{bmatrix} x_1^D & x_2^D & \dots & x_{n_D}^D \end{bmatrix}.$$

- **Queries** are computed using X_D:

$$Q = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_D} \end{bmatrix} = W_Q X_D.$$

- **Keys and values** are computed using X_E:

$$K = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_E} \end{bmatrix} = W_K X_E$$

$$V = \begin{bmatrix} v_1 & v_2 & \dots & v_{n_E} \end{bmatrix} = W_V X_E.$$

  - Weights using columnwise softmax (no mask):

$$Z = K^{\mathsf{T}} Q / \sqrt{d}, \qquad W = \text{softmax}(Z).$$

  - Weighted averages: $Y = VW$.

ENCODER-DECODER SELF-ATTENTION (3)

- Let us present equations in matrix form.
- Receives two inputs:

$$X_E = \begin{bmatrix} x_1^E & x_2^E & \ldots & x_{n_E}^E \end{bmatrix}$$

$$X_D = \begin{bmatrix} x_1^D & x_2^D & \ldots & x_{n_D}^D \end{bmatrix}.$$

- **Queries** are computed using X_D:

$$Q = \begin{bmatrix} q_1 & q_2 & \ldots & q_{n_D} \end{bmatrix} = W_Q X_D.$$

- **Keys and values** are computed using X_E:

$$K = \begin{bmatrix} q_1 & q_2 & \ldots & q_{n_E} \end{bmatrix} = W_K X_E$$

$$V = \begin{bmatrix} v_1 & v_2 & \ldots & v_{n_E} \end{bmatrix} = W_V X_E.$$

  - Weights using columnwise softmax (no mask):

$$Z = K^{\mathsf{T}} Q / \sqrt{d}, \qquad W = \mathrm{softmax}(Z).$$

  - Weighted averages: $Y = VW$.

Observations

- Number of outputs vectors is n_D.

ENCODER-DECODER SELF-ATTENTION (3)

- Let us present equations in matrix form.
- Receives two inputs:

$$X_E = \begin{bmatrix} x_1^E & x_2^E & \dots & x_{n_E}^E \end{bmatrix}$$

$$X_D = \begin{bmatrix} x_1^D & x_2^D & \dots & x_{n_D}^D \end{bmatrix}.$$

- **Queries** are computed using X_D:

$$Q = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_D} \end{bmatrix} = W_Q X_D.$$

- **Keys and values** are computed using X_E:

$$K = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_E} \end{bmatrix} = W_K X_E$$

$$V = \begin{bmatrix} v_1 & v_2 & \dots & v_{n_E} \end{bmatrix} = W_V X_E.$$

  - Weights using columnwise softmax (no mask):

$$Z = K^{\mathsf{T}} Q / \sqrt{d}, \qquad W = \mathrm{softmax}(Z).$$

  - Weighted averages: $Y = VW$.

Observations

- Number of outputs vectors is n_D.
- Value vectors determined by X_E.

ENCODER-DECODER SELF-ATTENTION (3)

- Let us present equations in matrix form.
- Receives two inputs:
$$X_E = \begin{bmatrix} x_1^E & x_2^E & \dots & x_{n_E}^E \end{bmatrix}$$
$$X_D = \begin{bmatrix} x_1^D & x_2^D & \dots & x_{n_D}^D \end{bmatrix}.$$
- **Queries** are computed using X_D:
$$Q = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_D} \end{bmatrix} = W_Q X_D.$$
- **Keys and values** are computed using X_E:
$$K = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_E} \end{bmatrix} = W_K X_E$$
$$V = \begin{bmatrix} v_1 & v_2 & \dots & v_{n_E} \end{bmatrix} = W_V X_E.$$
  - Weights using columnwise softmax (no mask):
$$Z = K^{\mathsf{T}} Q / \sqrt{d}, \qquad W = \mathrm{softmax}(Z).$$
  - Weighted averages: $Y = VW$.

Observations

- Number of outputs vectors is n_D.
- Value vectors determined by X_E.
- X_D only influences Y via W.

ENCODER-DECODER SELF-ATTENTION (3)

- Let us present equations in matrix form.
- Receives two inputs:

$$X_E = \begin{bmatrix} x_1^E & x_2^E & \dots & x_{n_E}^E \end{bmatrix}$$

$$X_D = \begin{bmatrix} x_1^D & x_2^D & \dots & x_{n_D}^D \end{bmatrix}.$$

- **Queries** are computed using X_D:

$$Q = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_D} \end{bmatrix} = W_Q X_D.$$

- **Keys and values** are computed using X_E:

$$K = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_E} \end{bmatrix} = W_K X_E$$

$$V = \begin{bmatrix} v_1 & v_2 & \dots & v_{n_E} \end{bmatrix} = W_V X_E.$$

  - Weights using columnwise softmax (no mask):

$$Z = K^{\mathrm{T}} Q / \sqrt{d}, \qquad W = \text{softmax}(Z).$$

  - Weighted averages: $Y = VW$.

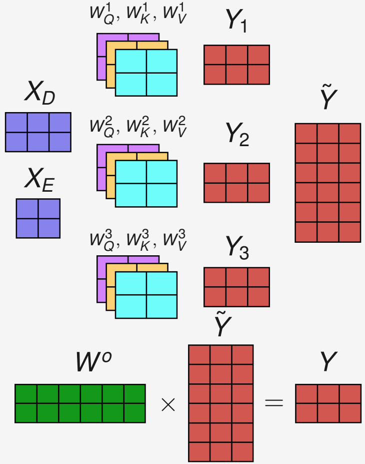
Observations

- Number of outputs vectors is n_D.
- Value vectors determined by X_E.
- X_D only influences Y via W.
- No masks needed: y_i still only depends on x_i^D and X_E.

ENCODER-DECODER MULTI-HEAD SELF-ATTENTION

- h parallel **encoder-decoder** self-attention blocks/heads.

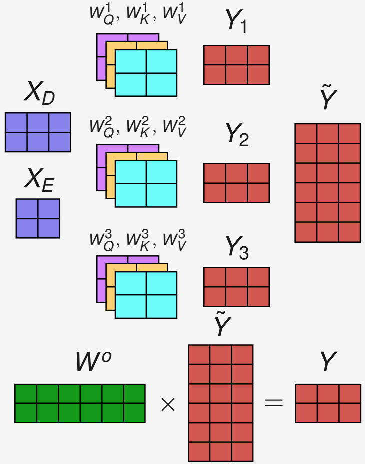
Example: $n_D = 3$, $d = 2$, $h = 3$



ENCODER-DECODER MULTI-HEAD SELF-ATTENTION

- h parallel **encoder-decoder** self-attention blocks/heads.
- Overall structure is identical to multi-head attention.

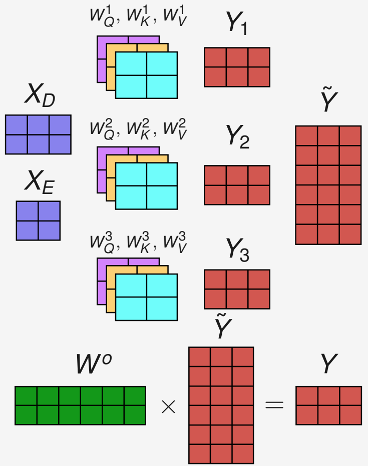
Example: $n_D = 3$, $d = 2$, $h = 3$



ENCODER-DECODER MULTI-HEAD SELF-ATTENTION

- h parallel **encoder-decoder** self-attention blocks/heads.
- Overall structure is identical to multi-head attention.
- **Difference**: when computing Y_i we use
  - X_D to compute queries $Q^i = W_Q^i X_D$,
  - X_E to compute keys $K^i = W_K^i X_E$ and values $V^i = W_V^i X_E$.

Example: $n_D = 3$, $d = 2$, $h = 3$



TRANSFORMER OVERVIEW