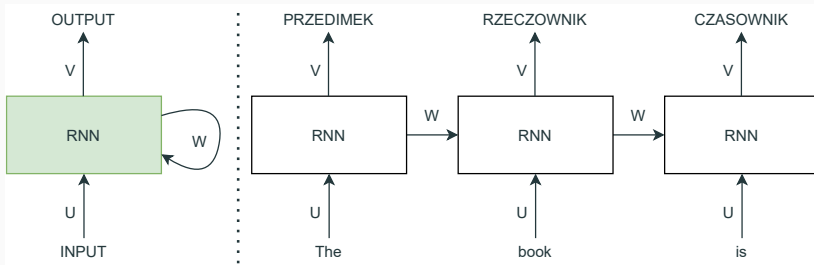


Poproszę o uwagę, czyli o atencji

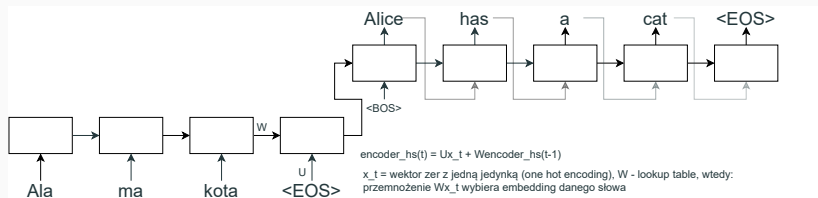
mgr inż. Dawid Wiśniewski - Przetwarzanie języka naturalnego

8 czerwca 2021

Atencja i



Sieci Seq2Seq (koder/dekoder) do automatycznego tłumaczenia:

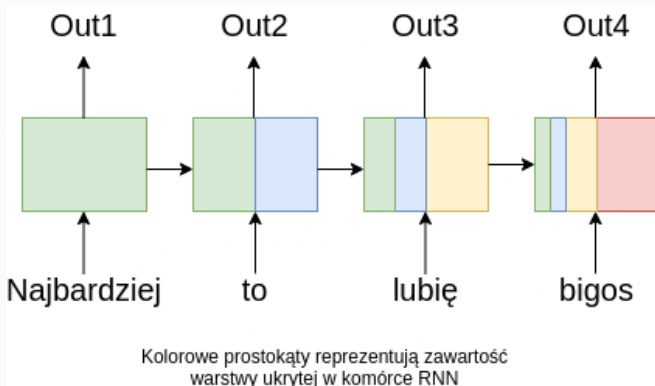


Prostokąty: komórki RNN generujące stany ukryte.

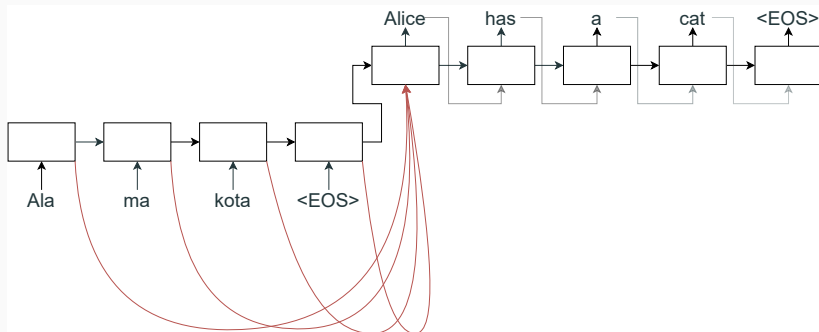
Cała (en)kodowana sekwencja, niezależnie od jej długości musi być ostatecznie osadzona w wektorze o niezmienniej długości.

Atencja iii

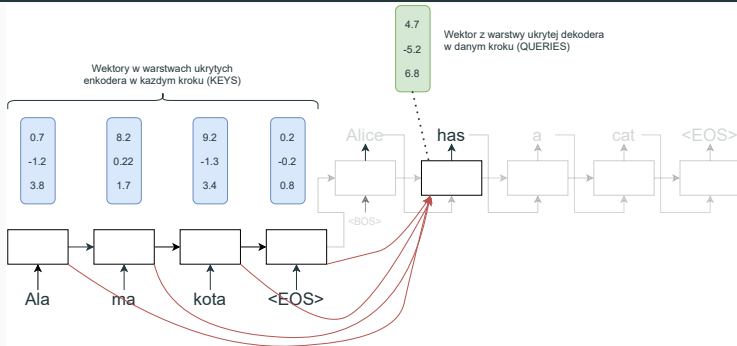
Czy cała historia słów rozkłada się po równo?



A gdyby tak dekodery widział wszystkie stany ukryte?



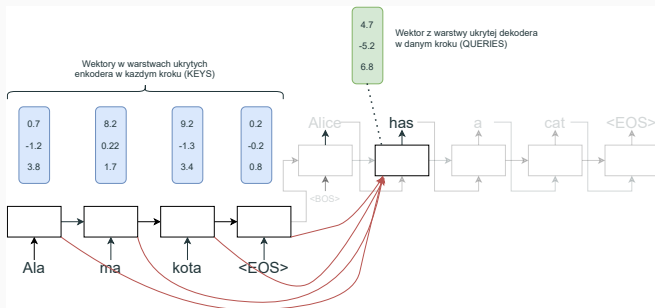
Atencja v



Mamy: sekwencję embeddingów z warstwy ukrytej każdego kroku (en)kodera (klucze): $\vec{e}_1, \vec{e}_2, \vec{e}_3, \dots, \vec{e}_n$

Mamy: embedding z warstwy ukrytej aktualnego kroku dekodera (zapytanie): \vec{d}_j

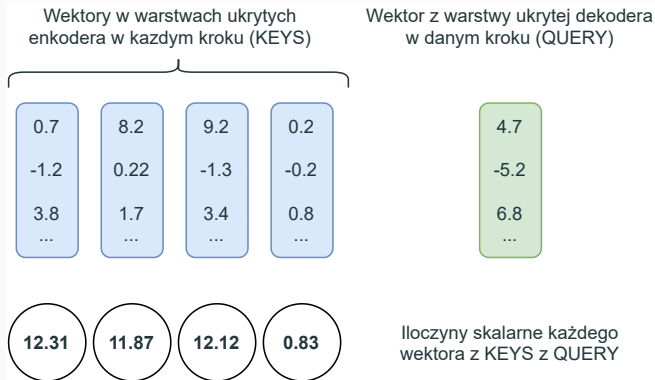
Atencja vi



Krok, liczymy iloczyn skalarny każdego stanu ukrytego (en)kodera z aktualnym stanem dekodera:

$a_{ij} = \vec{e}_i \cdot \vec{d}_j$, gdzie i to indeksy kolejnych tokenów w (en)koderze.

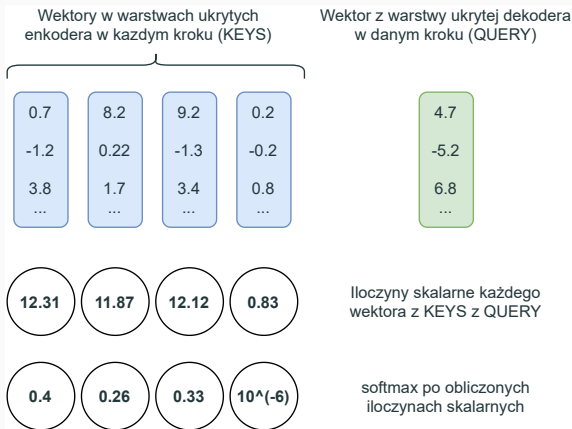
Atencja vii



Otrzymujemy wektor o długości takiej jak ilość elementów w (en)koderze.

Następnie wektor ten normalizujemy softmaxem.

Atencja viii

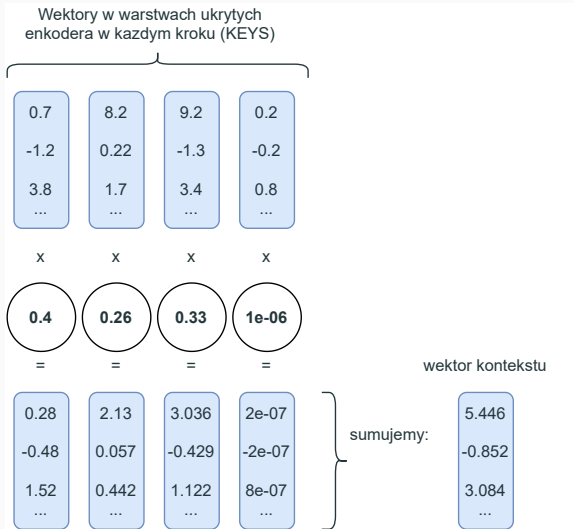


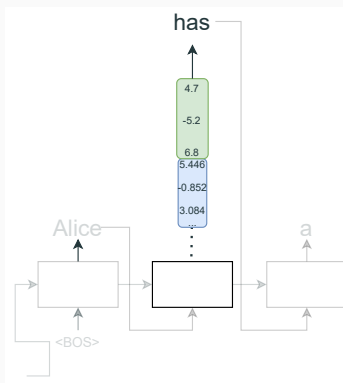
Otrzymując współczynniki mówiące jak ważne (podobne) jest dane słowo (z (en)kodera) wobec słowa z dekodera.

Wagi te użyte zostaną do policzenia wektora kontekstu, który obliczymy jako średnią ważoną (obliczonymi współczynnikami) po stanach ukrytych (en)kodera.

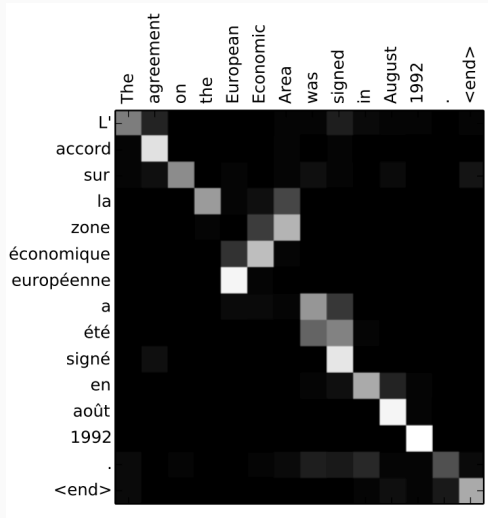
Wektor ten, zawiera informacje o istotnych tokenach pochodzących z enkodera. Możemy go np. skonkatelować ze stanem ukrytym dekodera, aby wygenerować token wyjściowy dla aktualnego kroku.

Atencja x





Stan ukryty z aktualnego kroku dekodera sklejany jest z wektorem kontekstu i przetwarzany przez warstwę liniową w celu wygenerowania (wybrania) tokenu wyjściowego.



Istnieje wiele podejść do atencji w NLP, ale wszystko zaczęło się od artykułu

Neural Machine Translation by Jointly Learning to Align and Translate
z 2014 roku:

<https://arxiv.org/abs/1409.0473>

Alternatywne formy atencji przedstawiono w 2015 roku w artykule
Effective Approaches to Attention-based Neural Machine Translation

<https://arxiv.org/abs/1508.04025>

Jednak nawet atencja pozostawia pewne problemy. Przede wszystkim, połączenia rekurencyjne wymuszają przetwarzanie danych sekwencyjnie.

Rok 2017 podarował nam przełom: architekturę transformer, opisaną w artykule:

Attention is all you need

<https://arxiv.org/abs/1706.03762>

Transformer ii

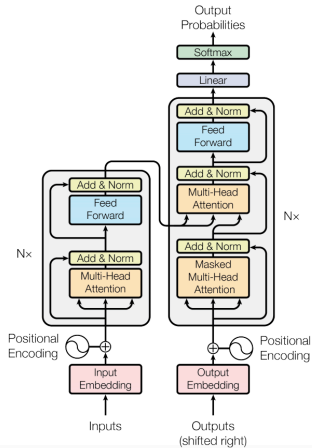
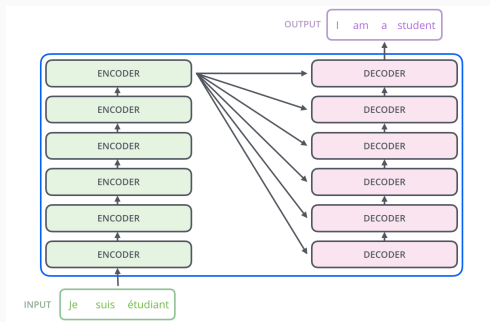


Figure 1: The Transformer - model architecture.

Obrazki na kolejnych slajdach pochodzą ze strony:
<https://jalammar.github.io/illustrated-transformer/>

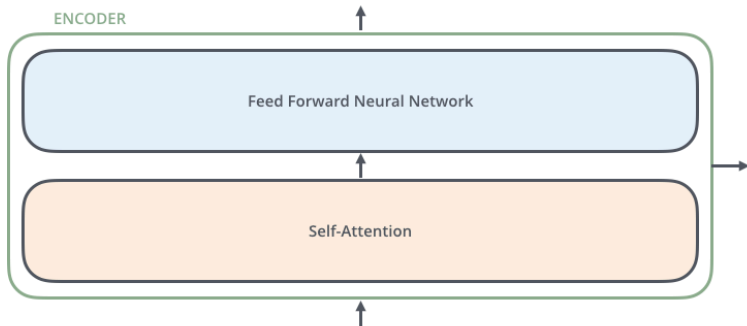
Transformer iv



Transformer składa się z 6 enkoderów i 6 dekodeków ustawionych jeden nad drugim.

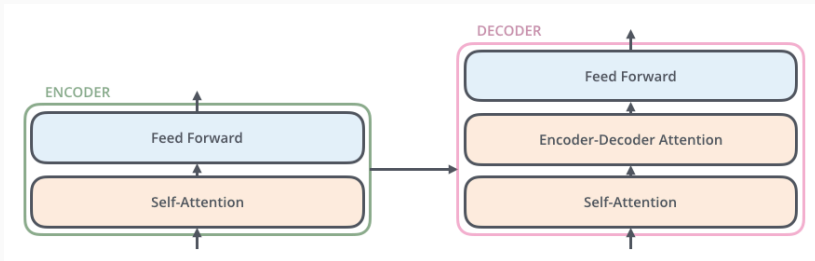
Każdy element sekwencji w ramach danego enkodera/dekodera może być przekształcany równolegle.

Transformer v



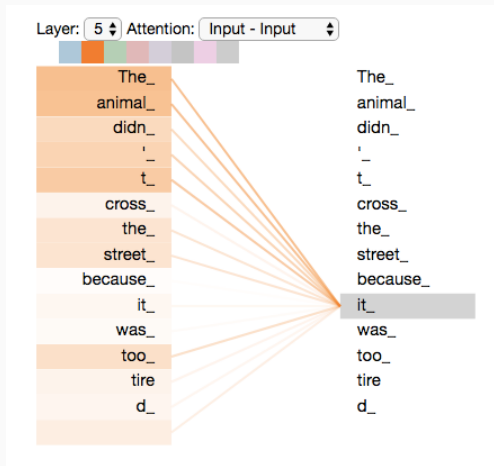
Każda warstwa enkodera składa się z mechanizmu samo-atencji, po której następuje warstwa feedforward (fully-connected).

Transformer vi

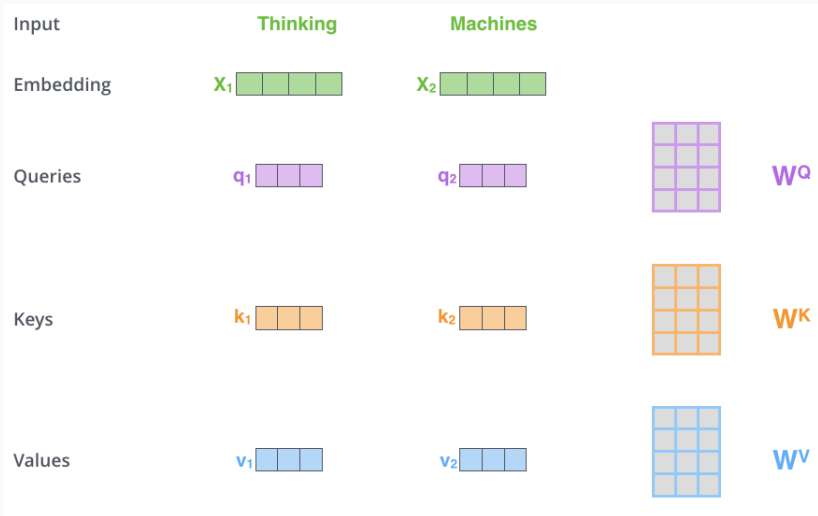


Dekoder natomiast wprowadza pośredni krok znanej już nam z początku wykładu uwagi między enkoderem a dekodorem.

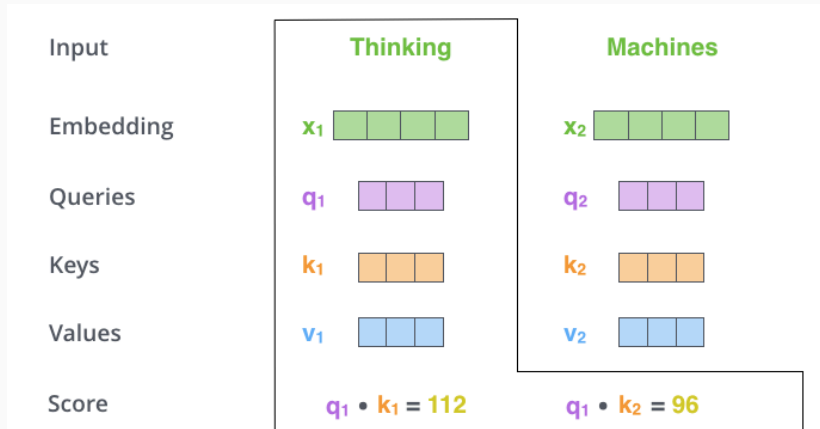
Transformer vii



Transformer viii



Transformer ix



Transformer x

Input

Embedding

Queries

Keys

Values

Score

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

Softmax

Thinking

x_1 

q_1 

k_1 

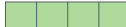
v_1 

$q_1 \cdot k_1 = 112$

14

0.88

Machines

x_2 

q_2 

k_2 

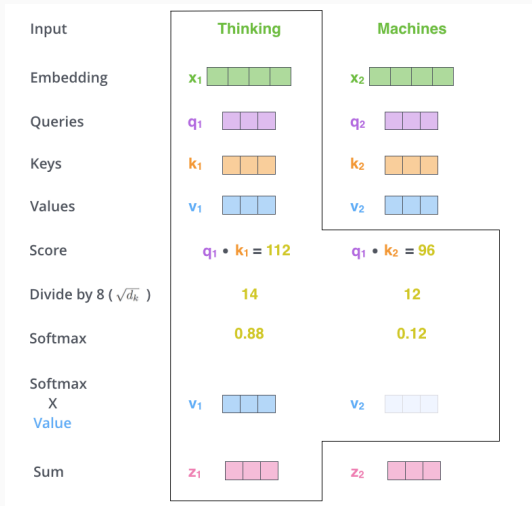
v_2 

$q_2 \cdot k_2 = 96$

12

0.12

Transformer xi



Transformer xii

$$\mathbf{X} \times \mathbf{W}^Q = \mathbf{Q}$$

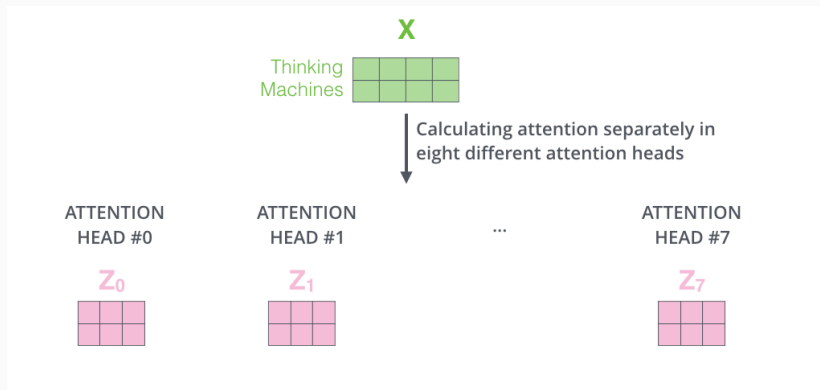

$$\mathbf{X} \times \mathbf{W}^K = \mathbf{K}$$


$$\mathbf{X} \times \mathbf{W}^V = \mathbf{V}$$


Transformer xiii

$$\text{softmax}\left(\frac{\begin{matrix} \text{Q} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix} \times \begin{matrix} \text{K}^T \\ \begin{array}{|c|c|} \hline \square & \square \\ \hline \square & \square \\ \hline \end{array} \end{matrix}\right) \begin{matrix} \text{V} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix}$$
$$= \begin{matrix} \text{Z} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix}$$

Transformer xiv



Transformer xv

1) Concatenate all the attention heads



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

x



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



Transformer xvi

1) This is our input sentence*

Thinking
Machines

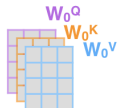
2) We embed each word*



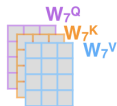
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



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



...



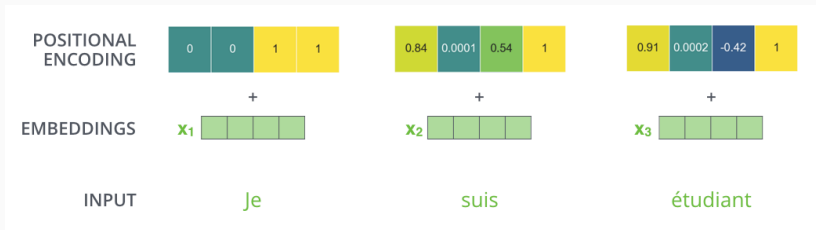
W^O



Z



Transformer xvii



Transformer xviii

Scaled Scores

0.7	0.1	0.1	0.1
0.1	0.6	0.2	0.1
0.1	0.3	0.6	0.1
0.1	0.3	0.3	0.3

+

Look-Ahead Mask

0	-inf	-inf	-inf
0	0	-inf	-inf
0	0	0	-inf
0	0	0	0

=

Masked Scores

0.7	-inf	-inf	-inf
0.1	0.6	-inf	-inf
0.1	0.3	0.6	-inf
0.1	0.3	0.3	0.3